Responzivnost Web Stranica

DANIJEL MARTINEK

Sadržaj

CSS RESPONZIVNOST

UVOD

Većina suvremenih web stranica ima responzivan dizajn. Danas je to nešto bez čega web stranica ne može postojati. Ali što je ustvari responzivan web dizajn? Responzivan web dizajn (RWD) pristup je u dizajniranju web stranica koji je zaslužan da ona izgleda dobro na svakom uređaju, odnosno da se može prilagoditi svakoj širini i rezoluciji ekrana. Nekada je responzivnost bila opcionalno svojstvo koje su dizajneri dodatno naplaćivali, ali uz sve veću razvijenost interneta i tehnologije (pogotovo pametnih telefona) to svojstvo jednostavno treba postojati u svakoj web stranici pogotovo ako uzmemo u obzir da sve više ljudi koristi mobitele za svakodnevno pretraživanje interneta. Tako dolazimo do pitanja kako postići tu responzivnost? Ima mnogo načina poput CSS Grid View metode ili Flexbox-a, ali ono što je zajednično većini su **media upiti** (eng. **media queries**).

CSS @media Rule

Media Rule ili media pravilo je skup media upita koji određuju različita CSS svojstva za različite širine zaslona i prikaza što se postiže pomoću **breakpoint**-a. **Media Breakpoints (Media točke prekida)** su točke kojima se određuje raspon, minimalna ili maksimalna širina zaslona za koje će se određena CSS svojstva prikazivati te se određuju uglavnom u pikselima (px).

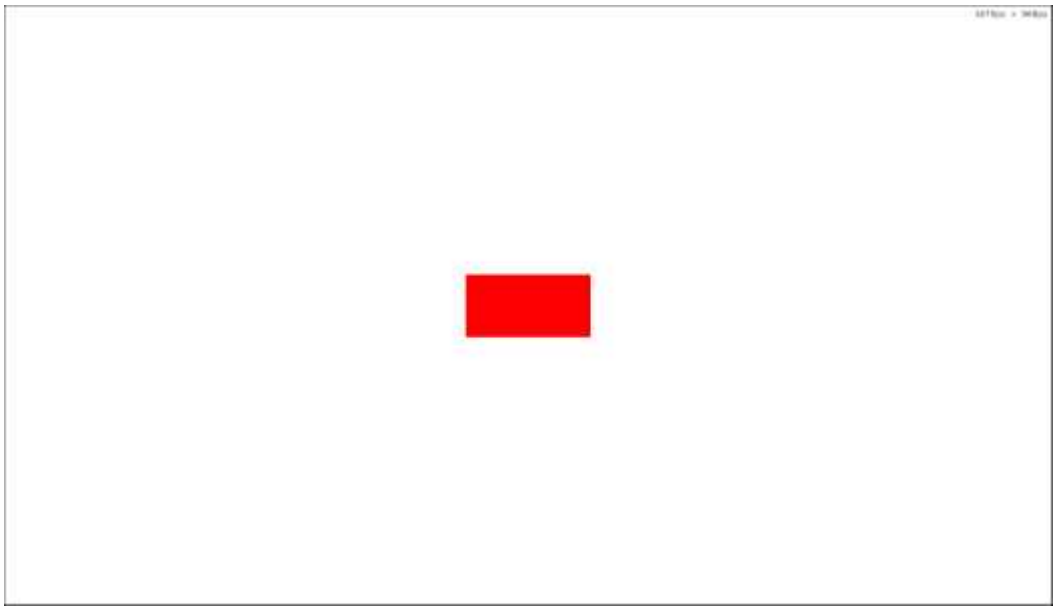
Primjer:

css kod

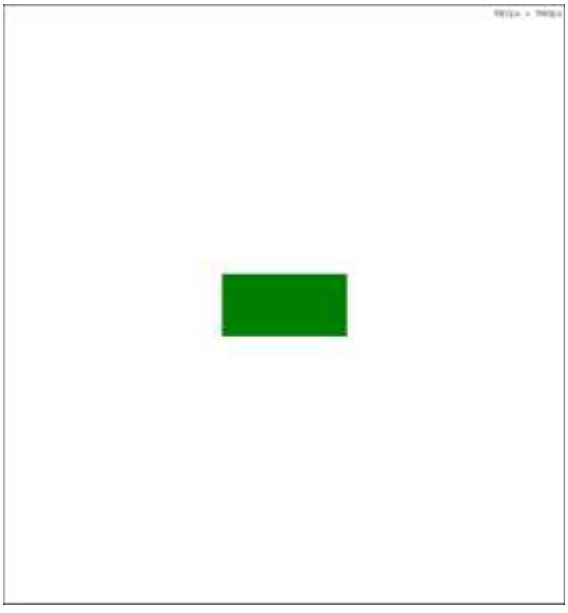
```
.mediaDiv {
        width: 200px;
        height: 100px;
        margin: auto;
        position: absolute;
        top: 0; left: 0; bottom: 0; right: 0;

        background-color: blue;
}

@media only screen and (min-width: 600px) {
        .mediaDiv{
                background-color: green;
        }
}

@media only screen and (min-width: 1200px) {
        .mediaDiv{
                background-color: red;
        }
}
```

Rezultat:

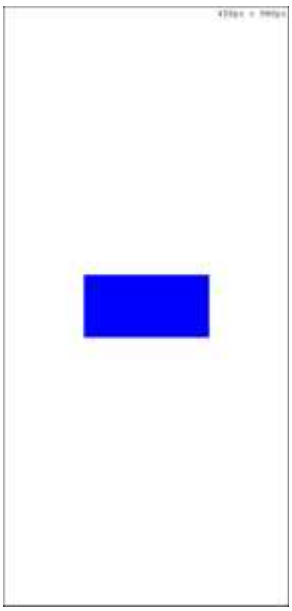
1. Iznad 1200px širine



2. Između 600px i 1200px širine



3. Ispod 600px širine



U ovom primjeru svi media upiti su zadani pomoću *min-width* varijable što ujedno znači da je primjer rađen "*Mobile First*" pristupom – osnovna svojstva za zaslone manje od 600px širine ujedno su i početna svojstva.
Mobile First pristup osigurava relativno dobar prikaz web stranice kod slučaja greške ili nedefiniranog media uvjeta na određenom rasponu širine zaslona. Mnogi dizajneri koriste *Mobile First* pristup i zbog toga što se mobilna tehnologija iznimno brzo razvija.

CSS responzivnost

Osnovne tehnike za responzivnost web stranica*:

- CSS grid view
- CSS grid layout (predstavljen u CSS3 verziji)
- Flexbox

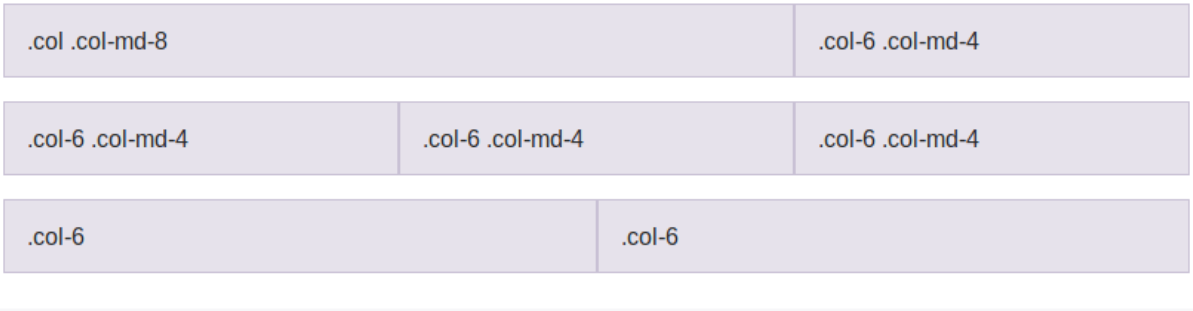
*Iznad navedene tehnike su tehnike koje se najčešće koriste i odnose se globalno na cijelu web stranicu što znači da ih se može upotrebljavati na svakom elementu te stranice.

CSS grid view

Najpoznatija i najkorištenija metoda responzivnog dizajna koju koriste Boostrap i Material UI – dva najpoznatija *CSS component frameworka*, zapravo je “hakiranje” CSS-a. Zašto je to tako?

Razlog je taj što u CSS-u **ne postoji** ni jedno svojstvo direktno određeno za korištenje pri CSS grid view metodi, već je to spoj više svojstava koje nam CSS donosi. Zašto je onda toliko korišteno u svijetu web responzivnosti?

Jedan od glavnih razloga je podržavanje preglednika (*Browser Support*). Upravo zbog toga što je *grid view* spoj više nepovezanih svojstava, koja su ujedno i osnovna svojstva CSS-a, podržan je u svim preglednicima svih verzija, od starijih do najnovijih, što ga čini idealnim izborom za spomenute *component frameworke.*



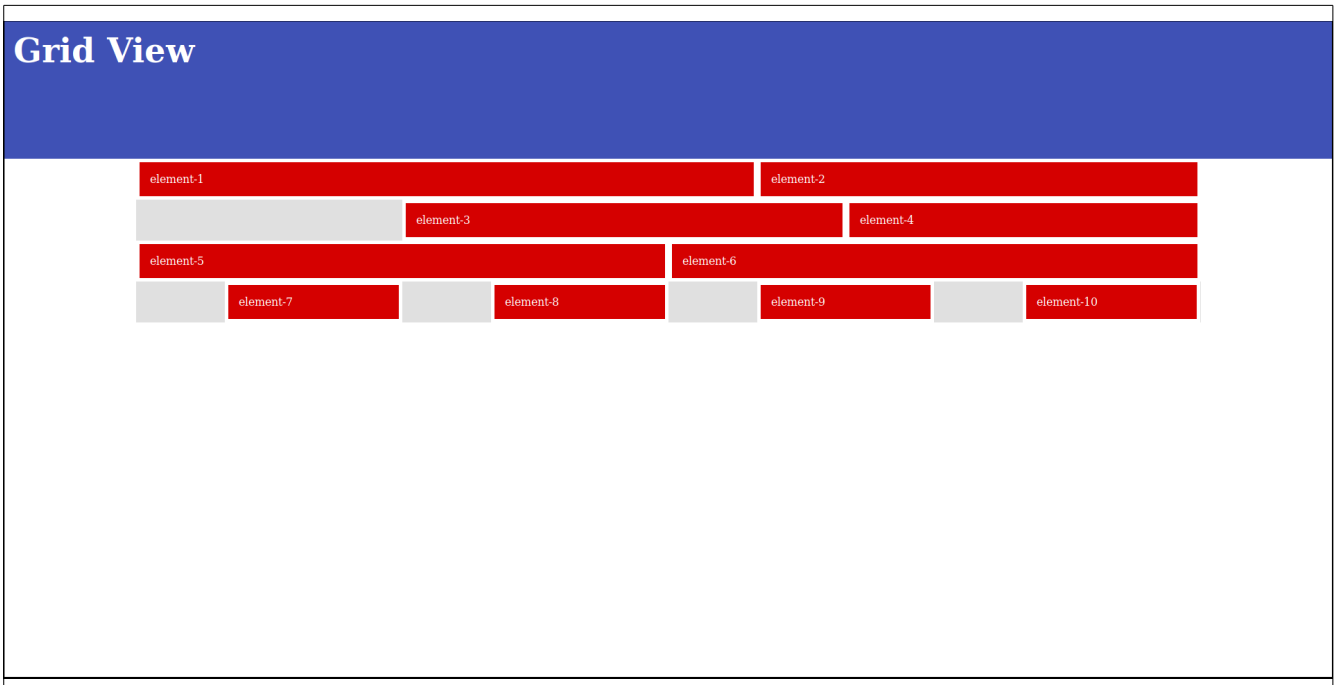

```
<!-- Stack the columns on mobile by making one full-width and the other half-width -->
<div class="row">
  <div class="col col-md-8">.col .col-md-8</div>
  <div class="col-6 col-md-4">.col-6 .col-md-4</div>
</div>

<!-- Columns start at 50% wide on mobile and bump up to 33.3% wide on desktop -->
<div class="row">
  <div class="col-6 col-md-4">.col-6 .col-md-4</div>
  <div class="col-6 col-md-4">.col-6 .col-md-4</div>
  <div class="col-6 col-md-4">.col-6 .col-md-4</div>
</div>

<!-- Columns are always 50% wide, on mobile and desktop -->
<div class="row">
  <div class="col-6">.col-6</div>
  <div class="col-6">.col-6</div>
</div>
```

Primjer Bootstrap grid sistema

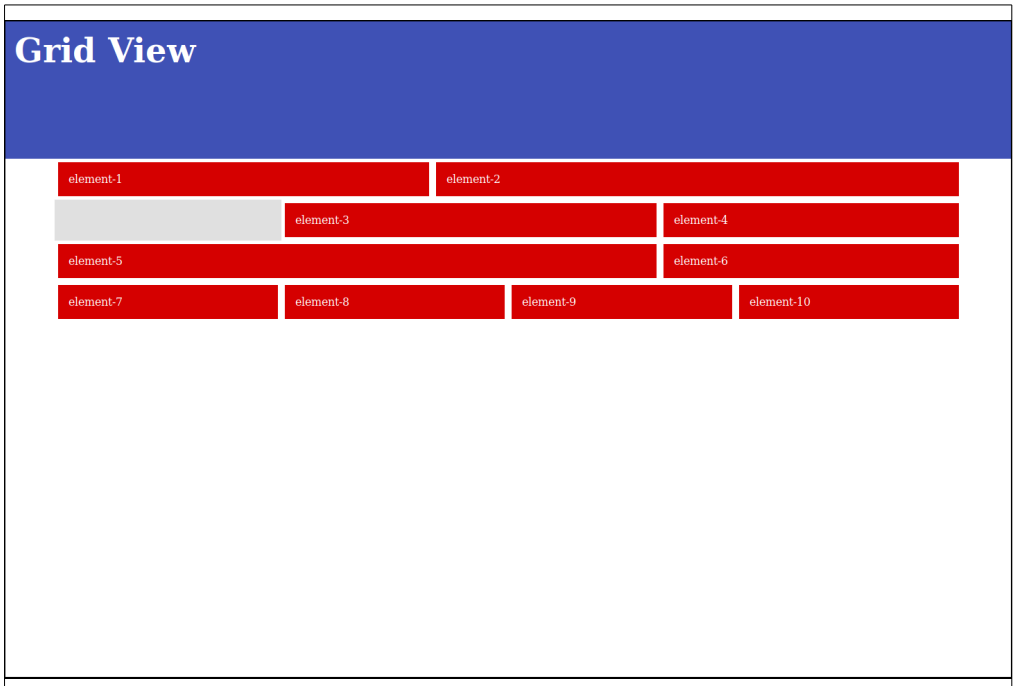
Drugi glavni razlog je jednostavnost primjene. Naime, njegovo korištenje ne zahtjeva pisanje klasa (*class*) za svaki pojedini element nego se jedna klasa može primjeniti na neodređeni broj elemenata. Upravo je zbog toga odlično i elegantno rješenje kod većih web stranica koje sadrže velik broj blokovnih elemenata.

Isto tako promjene dimenzija elementa se mogu vršiti u samom HTML dokumentu unutar *class* atributa bez potrebe mjenjanja klase unutar CSS dokumenta. Ovo je ujedno i mana ove tehnike jer su u CSS dokumentu unaprijed određene sve klase za pojedinu širinu elementa u odnosu na određenu širinu zaslona od kojih se velik dio uopće ne koristi. Zbog toga je poželjno koristiti što manje media uvjeta odnosno točaka prekida, no samim time se ujedno gubi i na fleksibilnosti ove metode.

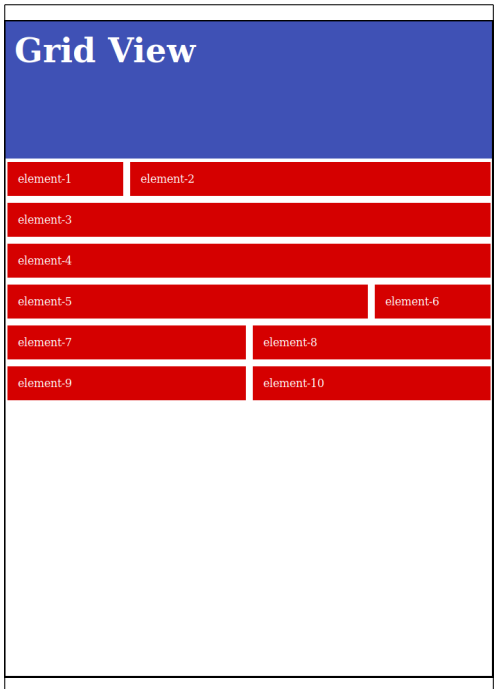
Još jedna mana ove metode je pozicioniranje elemenata jednakim redoslijedom na svim širinama i vrstama zaslona.



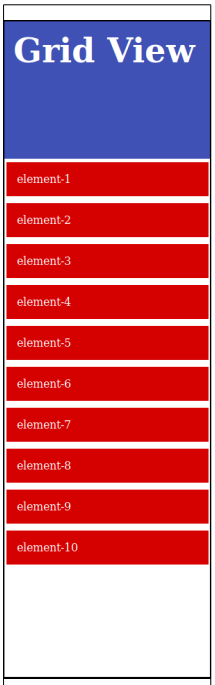
Prikaz 1: Velika širina zaslona (iznad 1610px)



Prikaz 2: Srednja širina zaslona (između 710px i 1610px)



Prikaz 3: Mala širina zaslona (između 320px i 710px)



Prikaz 4: Mobilni prikaz (ispod 320px)

Napomena: Točke prekida navedene u ovom primjeru odnose se samo na taj primjer. One su proizvoljne te nije određen njihov broj kao ni raspon između njih, osim ako se ne radi o *component frameworku*.

U navedenom primjeru možemo uočiti kako se kod smanjivanja širine zaslona redoslijed elemenata ne mijenja, ali se mijenja opći prikaz web stranice.

Način rada – CSS grid view

Prije svega trebamo odrediti način prikaza web stranice (`display` svojstvo). U ovom slučaju odabrat ćemo `display: table;` . Time dobivamo da se elementi unutar klase ponašaju poput elemenata tablice – možemo koristiti retke i kontrolirati širinu svakog elementa u pojedinom retku.

```
* {
	margin: 0;
	padding: 0;
}

.row::after {
	display: table;
}
```

CSS grid view radi pomoću relativne širine elementa u odnosu na *parent* element. Drugim riječima, ako imamo element koji zauzima 2/3 zaslona određujemo mu `width: 66.66%;` što je 2/3 od širine *parent* elementa u ovom slučaju *body* stranice.

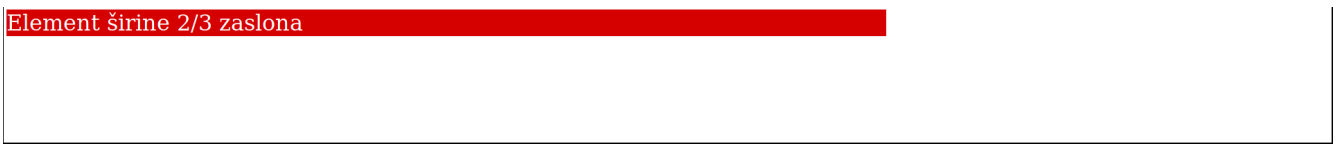
style.css

```
* {
	margin: 0;
	padding: 0;
}

.row::after {
	display: table;
}

.elementDiv{
	width: 66.66%;
}


.grid-item{
	background-color: #d50000;
	color: #ffffff;
	border: 5px solid #ffffff;
	font-size: 2em;
}
```

index.html

```
<!DOCTYPE html>
<html>
<head>

<title>CSS Grid View</title>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">
<!-- **viewport meta atribut daje html stranici instrukcije kako kontrolirati dimenzije stranice te smanjivanje/povećavanje** -->

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

</head>
<body>

<div class="row">
        <div class="elementDiv grid-item">
                Element širine 2/3 zaslona
        </div>
</div>

</body>
</html>
```

Rezultat:



Povećanjem ili smanjenjem širine zaslona zadani element ostat će u jednakom omjeru u odnosu na *parent* element (*body*).

Da bismo postavili više od jednog elementa u jedan redak, trebamo dodati `float: left;` svojstvo. Zbog toga dodajemo .col klasu koja će se odnositi jednako na svaki pojedini element.

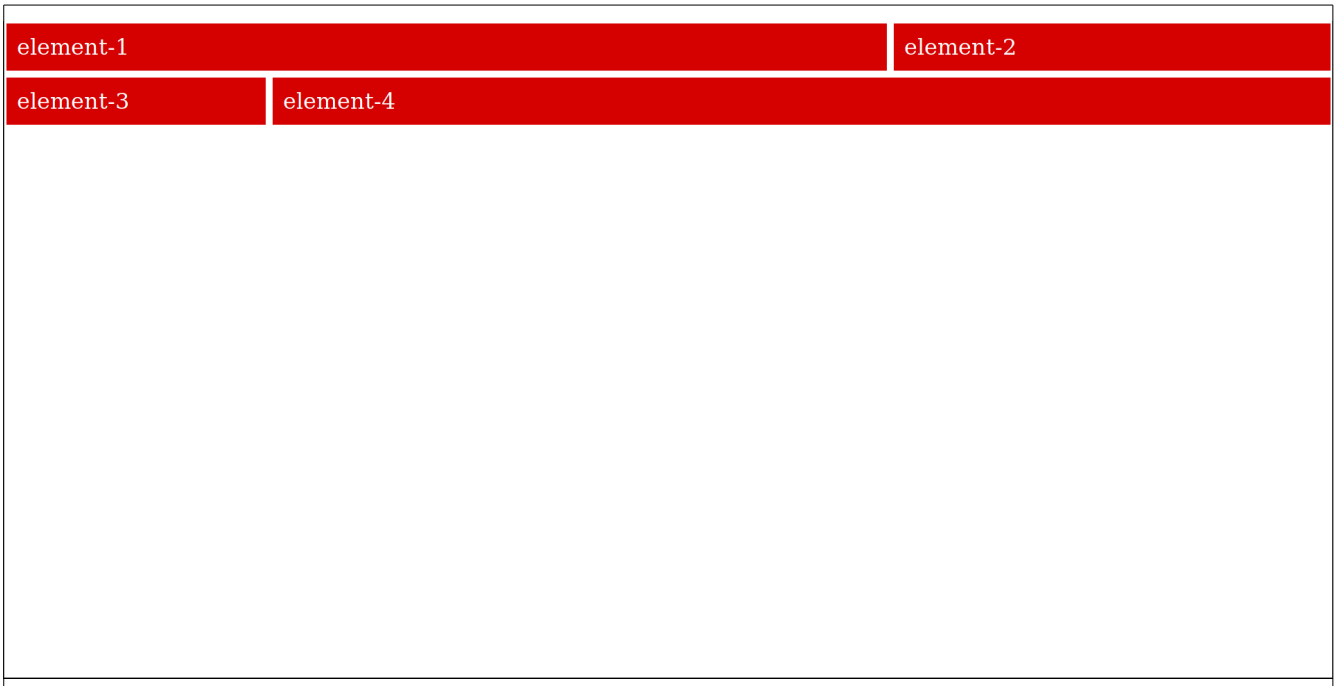
style.css sada izgleda ovako

```
* {
	margin: 0;
	padding: 0;
}

.row::after {
	display: table;
}

.col {
	float: left;
}

.element-1{
	width: 66.66%;
}

.element-2{
	width: 33.33%;
}

.element-3{
	width: 20%;
}

.element-4{
	width: 80%;
}


.grid-item{
	background-color: #d50000;
	color: #ffffff;
	border: 5px solid #ffffff;
	font-size: 2em;
	padding: 15px;
}
```

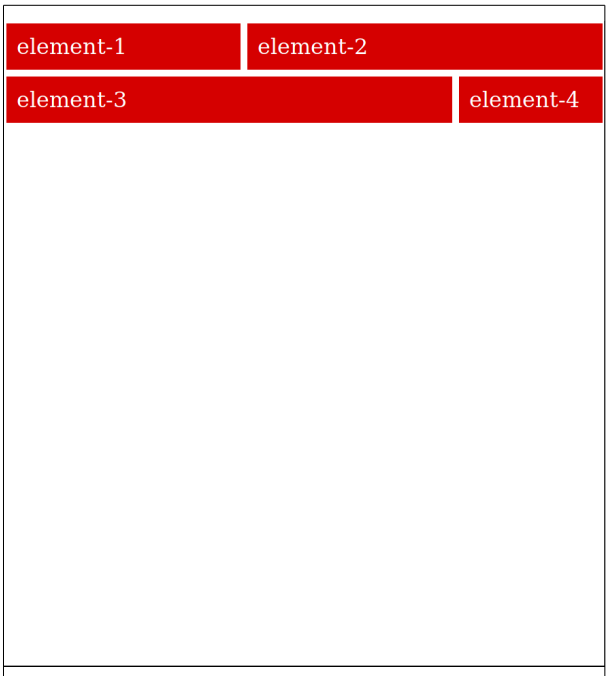
Nadalje, ukoliko pridodamo media točke prekida, možemo direktno kontrolirati koje će širine određeni element biti za pojedini media raspon.

```
@media only screen and (max-width: 600px) {
	.element-1{
		width: 100%;
	}
	.element-2{
		width: 100%;
	}
	.element-3{
		width: 50%;
	}
	.element-4{
		width: 50%;
	}
}

@media only screen and (min-width: 600px) and (max-width: 1200px) {
	.element-1{
		width: 40%;
	}
	.element-2{
		width: 60%;
	}
	.element-3{
		width: 75%;
	}
	.element-4{
		width: 25%;
	}
}
```

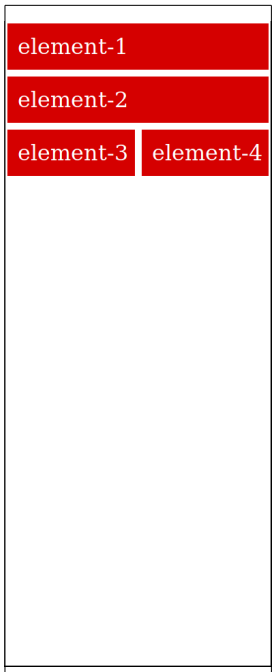
Rezultat:



Prikaz elemenata na zaslonu većem od 1200px



Prikaz elemenata na zaslonu veličine između 600px i 1200px



Prikaz elemenata na zaslonu manjem od 600px

Iz primjera se vidi koliko je jednostavno upravljati CSS grid view metodom te koliko je ona zapravo fleksibilna u smislu izrade responzivnog grida za više elemenata.

Velika mana širine elementa pomoću relativnog odnosa na *parent* element je teško postavljanje elementa fiksne širine za svaku vrstu i širinu zaslona te zahtjeva prepravak i podešavanje css koda.

Ako proširimo media uvjete i podijelimo redak na 12 segmenata, dobijemo *CSS grid view toolkit* koji možemo koristiti bilogdje.

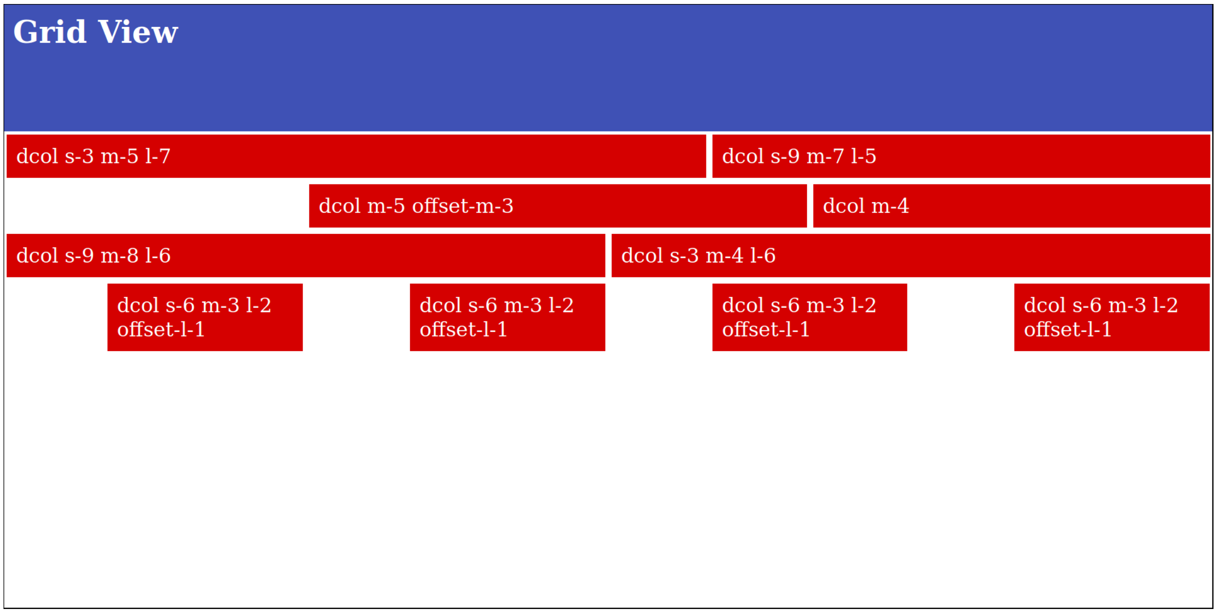
Dodane klase unutar media uvjeta u style.css

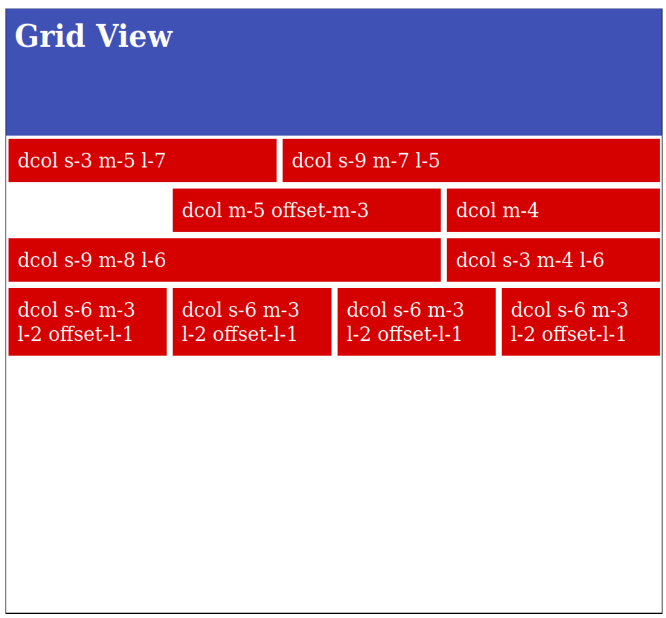
```
@media only screen and (min-width: 320px) {
	.s-1 {width: 8.33%;}
	.s-2 {width: 16.66%;}
	.s-3 {width: 25%;}
	.s-4 {width: 33.33%;}
	.s-5 {width: 41.66%;}
	.s-6 {width: 50%;}
	.s-7 {width: 58.33%;}
	.s-8 {width: 66.66%;}
	.s-9 {width: 75%;}
	.s-10 {width: 83.33%;}
	.s-11 {width: 91.66%;}
	.s-12 {width: 100%;}
}
@media only screen and (min-width: 710px) {
	.m-1 {width: 8.33%;}
	.m-2 {width: 16.66%;}
	.m-3 {width: 25%;}
	.m-4 {width: 33.33%;}
	.m-5 {width: 41.66%;}
	.m-6 {width: 50%;}
	.m-7 {width: 58.33%;}
	.m-8 {width: 66.66%;}
	.m-9 {width: 75%;}
	.m-10 {width: 83.33%;}
	.m-11 {width: 91.66%;}
	.m-12 {width: 100%;}
}
@media only screen and (min-width: 1610px) {
	.l-1 {width: 8.33%;}
	.l-2 {width: 16.66%;}
	.l-3 {width: 25%;}
	.l-4 {width: 33.33%;}
	.l-5 {width: 41.66%;}
	.l-6 {width: 50%;}
	.l-7 {width: 58.33%;}
	.l-8 {width: 66.66%;}
	.l-9 {width: 75%;}
	.l-10 {width: 83.33%;}
	.l-11 {width: 91.66%;}
	.l-12 {width: 100%;}
}
```

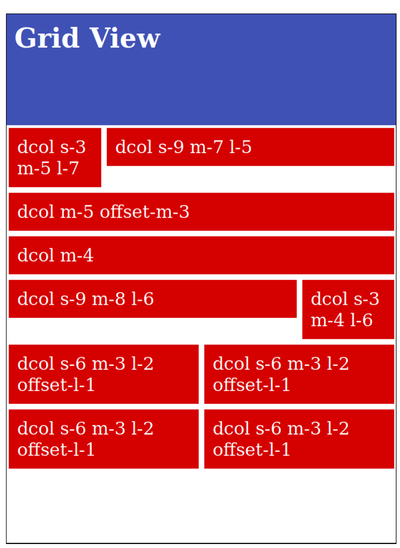
Elementi u index.html

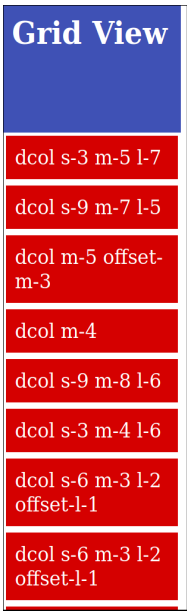
```
<div class="container">
	<div class="row">
		<div class="col s-3 m-5 l-7 grid-item">
			dcol s-3 m-5 l-7
		</div>
		<div class="col s-9 m-7 l-5 grid-item">
			dcol s-9 m-7 l-5
		</div>
	</div>
	<div class="row">
		<div class="col m-5 offset-m-3 grid-item">
			dcol m-5 offset-m-3
		</div>
		<div class="col m-4 grid-item">
			dcol m-4
		</div>
	</div>
	<div class="row">
		<div class="col s-9 m-8 l-6 grid-item">
			dcol s-9 m-8 l-6
		</div>
		<div class="col s-3 m-4 l-6 grid-item">
			dcol s-3 m-4 l-6
		</div>
	</div>
	<div class="row">
		<div class="col s-6 m-3 l-2 offset-l-1 grid-item">
			dcol s-6 m-3 l-2 offset-l-1
		</div>
		<div class="col s-6 m-3 l-2 offset-l-1 grid-item">
			dcol s-6 m-3 l-2 offset-l-1
		</div>
		<div class="col s-6 m-3 l-2 offset-l-1 grid-item">
			dcol s-6 m-3 l-2 offset-l-1
		</div>
		<div class="col s-6 m-3 l-2 offset-l-1 grid-item">
			dcol s-6 m-3 l-2 offset-l-1
		</div>
	</div>
</div>
```

Rezultat:


Grid View
dcol s-3 m-5 l-7
dcol s-9 m-7 l-5
dcol m-5 offset-m-3
dcol m-4
dcol s-9 m-8 l-6
dcol s-3 m-4 l-6
dcol s-6 m-3 l-2 offset-l-1
dcol s-6 m-3 l-2 offset-l-1
dcol s-6 m-3 l-2 offset-l-1
dcol s-6 m-3 l-2 offset-l-1


Grid View
dcol s-3 m-5 l-7
dcol s-9 m-7 l-5
dcol m-5 offset-m-3
dcol m-4
dcol s-9 m-8 l-6
dcol s-3 m-4 l-6
dcol s-6 m-3 l-2 offset-l-1
dcol s-6 m-3 l-2 offset-l-1
dcol s-6 m-3 l-2 offset-l-1
dcol s-6 m-3 l-2 offset-l-1


Grid View
dcol s-3 m-5 l-7
dcol s-9 m-7 l-5
dcol m-5 offset-m-3
dcol m-4
dcol s-9 m-8 l-6
dcol s-3 m-4 l-6
dcol s-6 m-3 l-2 offset-l-1
dcol s-6 m-3 l-2 offset-l-1
dcol s-6 m-3 l-2 offset-l-1
dcol s-6 m-3 l-2 offset-l-1



Osim opcija širine, na ovom primjeru možemo vidjeti i opcije pozicioniranja elemenata (*offset*).

Pozicioniranje postižemo na jednak način kao i širinu, samo što umjesto `width` svojstva unutar media uvjeta pišemo `margin-left` svojstvo s obzirom na to da je svaki element definiran da pluta na lijevo (`float: left;`).

ANALIZA *CSS grid view* metode

PREDNOSTI:

- podrška svih vrsta i verzija preglednika
- jednostavnost primjene
- definiranje širine elementa po relativnom odnosu na *parent* element (u %)
- jednostavno definiranje *nested* grida
- fleksibilnost – jedna klasa se može upotrebljavati neodređeni broj puta
- podrška Boostrapa i Material dizajna

MANE:

- “nepostojeća” css svojstva
- velik broj css klasa koja se ne koriste
- elementi fiksne širine zahtjevaju prepravak i podešavanje css koda

Zaključak:

CSS grid view je s nama gotovo od pojave pametnih telefona i pokazao se kao pouzdana tehnika u izradi responzivnih web stranica jednostavnim načinom primjene. Iako ima svojih mana, CSS grid view danas je vrlo popularan i, uz flexbox, ostat će popularan sve dok se najveća imena u css component developmentu poput Boostrapa ne odluče na neko drugo rješenje za responzivnost web stranica.

CSS grid layout

Iako ga mnogi zamjenjuju s CSS grid view-om, to su dvije potpuno različite tehnike web responzivnosti.

CSS grid layout metoda se, za razliku CSS grid view metode, odnose na točno određena svojstva CSS-a namjenjena samo za izradu responzivnih web stranica. Riječ u sredini oba naziva, “grid”, u ovom slučaju se odnosi na stvarni grid sistem, a ne na tablični prikaz koji je slučaj kod CSS grid view-a pa se nekad naziva samo *“CSS grid”* što zbunjuje ljude.

CSS grid layout je nova metoda koja još nije pod W3C standardom, iako je kandidat te zbog toga podržanost web pregledinka nije potpuna. No zbog svoje jednostavnosti korištenja sve više dizajnera počinje raditi responzivne stranice ovom metodom. Valjda napomenuti da najpopularniji web preglednici poput Google Chromea, Mozille Firefox i Safaria podržavaju CSS grid layout iako još nije W3C standard.

Velika prednost CSS grid layouta je definiranje grida i polja grida bilo koje visine i širine te se određeni element može provlačiti kroz više susjednih polja kao što vidimo u primjeru iznad.

Način rada - CSS grid layout

HTML grida sastoji se od parent elementa unutar kojeg se nalaze elementi prikaza rapoređenih po gridu.

index.html

```
<!DOCTYPE html>
<html>
<head>

<title>CSS grid layout</title>

<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">

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

</head>
<body>

<div class="wrapper">
        <div class="header"> HEADER </div>
        <div class="footer"> FOOTER </div>
        <div class="navbar"> NAVBAR </div>
        <div class="content"> CONTENT </div>
        <div class="ads"> ADVERTISING </div>
</div>

</body>
</html>
```

style.css

```
* {box-sizing: border-box;}

.wrapper {
	max-width: 1400px;
	margin: 0 auto;
}

.wrapper > div {
	border: 1px solid rgb(133,100,150);
	background-color: rgba(133,231,88,.5);
	padding: 1em;
	color: green;
}

.wrapper {
	display: grid;
	grid-template-columns: repeat(3, 1fr);
	grid-gap: 5px;
	grid-auto-rows: minmax(100px, auto);
}
.header {
	grid-column: 1 / 4;
	grid-row: 1 / 3;
}
.navbar {
	grid-column: 1 / 3;
	grid-row: 2 / 3;
}
.content {
	grid-column: 1 / 3;
	grid-row: 3 / 7;
}
.footer {
	grid-column: 1 / 5;
	grid-row: 7 / 8;
}
.ads {
	grid-column: 3 / 5;
	grid-row: 3 / 5;
}
```

Rezultat:

Ovdje vidimo da raspored elemenata nije jednak kao u html dokumentu što je bio slučaj kod CSS grid view-a, već je on definiran u samom css-u. To nam omogućava različito pozicioniranje elemenata za različite širine zaslona. Isto tako vidimo da se elementi mogu preklapati, odnosno pozicionirati jedan na drugi.

```
.wrapper {
	display: grid;
	grid-template-columns: repeat(3, 1fr);
	grid-gap: 5px;
	grid-auto-rows: minmax(100px, auto);
}
```

Unutar *wrapper-a*, podešen je prikaz za grid - `display: grid;` to nam omogućava definiranje grid polja.

Veličina grid polja se definira pomoću `grid-template-columns` i `grid-template-rows` svojstva ako imamo konačan broj polja. Definirati možemo pomoću *repeat* atributa tako da odredimo `grid-template-columns: repeat(3, 1fr);` gdje "3" označava broj ponavljanja, odnosno broj polja u retku, a "1fr" je predložak za svako ponavljanje i odnosi se na odnos veličine prema ostalim poljima retka. Drugim riječima, redak se sastoji od 3 polja iste širine. Isto tako pomoću "fr" (fragmenta) možemo definirati pojedinačna polja po flexbox metodi - `template-columns: 1fr 2fr 1fr 1fr;` gdje će drugo polje biti dvostruko veće od ostalih polja.

Definirati polja možemo i pomoću osnovnih jedinica poput pixela, em-a, postotka…

Ukoliko nemamo konačan broj redaka ili stupaca, koristimo `grid-auto-columns` ili `grid-auto-rows`. Na taj način polja će se automatski dodati u grid kada ih definiramo unutar css-a.

Za stupce i retke također možemo postaviti minimalnu ili/i maksimalnu visinu/širinu – `grid-auto-rows: minmax(100px, 500px);`

grid-gap svojstvo označava *gutter*, odnosno razmak između dva polja – vertikalno i horizontalno.

Razmak možemo podesiti pojedinačno za redak (`grid-row-gap`) i pojedinačno za stupac (`grid-column-gap`)

Kada smo definirali wrapper, tj. okvir grida i polja grida, moramo definirati područje koje zauzima pojedini element.

Primjer:

```
.content {
        grid-column: 1 / 3;
        grid-row: 3 / 7;
}

.footer {
        grid-column: 1 / 5;
        grid-row: 7 / 8;
}
```

Na ovaj način određujemo rubove raspona elementa te kroz koja polja prolaze.

Kod `grid-column: 1 / 3;` “1” označava lijevi rub prvog polja grida, a “3” desni rub drugog polja grida u istom retku što znači da element prolazi kroz prva dva stupca. Nadalje imamo `grid-row: 3 / 7;` gdje se “3” odnosi na početak trećeg ruba stupca, a “7” na krajnji rub šestog stupca. Povezivanjem ova dva svojstva dobivamo *template* area, tj. područje prolaska elementa.

Drugi način određivanja raspona je pomoću `grid-template-areas` svojstva koje definiramo unutar *wrapper* klase.

```
.wrapper {
    grid-template-columns: 1fr 3fr;
    grid-template-areas:
      "header header  header"
      "nav    content sidebar"
      "nav    content ad"
      "footer footer  footer";

}
```

Svaka riječ označava jedno polje, a svi elementi unutar jednih navodnika označavaju redak.

Na taj način svakom polju dodjeljujemo naziv preko kojeg ćemo povezati elemente.

```
.main-head {
  grid-area: header;
}
.content {
  grid-area: content;
}
.main-nav {
  grid-area: nav;
}
.side {
  grid-area: sidebar;
}
.ad {
  grid-area: ad;
}
.main-footer {
  grid-area: footer;
}
```

Ovaj pristup je idealan za responzivni dizajn jer se unutar pojedinog media uvjeta mjenja samo jedna osnovna klasa – *wrapper*.

```
@media (min-width: 700px) {
  .wrapper {
    grid-template-columns: 1fr 4fr 1fr;
    grid-template-areas:
      "header  header"
      "nav     nav"
      "sidebar content"
      "ad      footer";

   }
}
```

Na taj način jednostavno možemo kontrolirati, ne samo veličinu elementa, nego i poziciju elementa za pojedinu širinu zaslona.

Dodamo li nekoliko media uvjeta dobit ćemo potpuno responzivnu web stranicu:

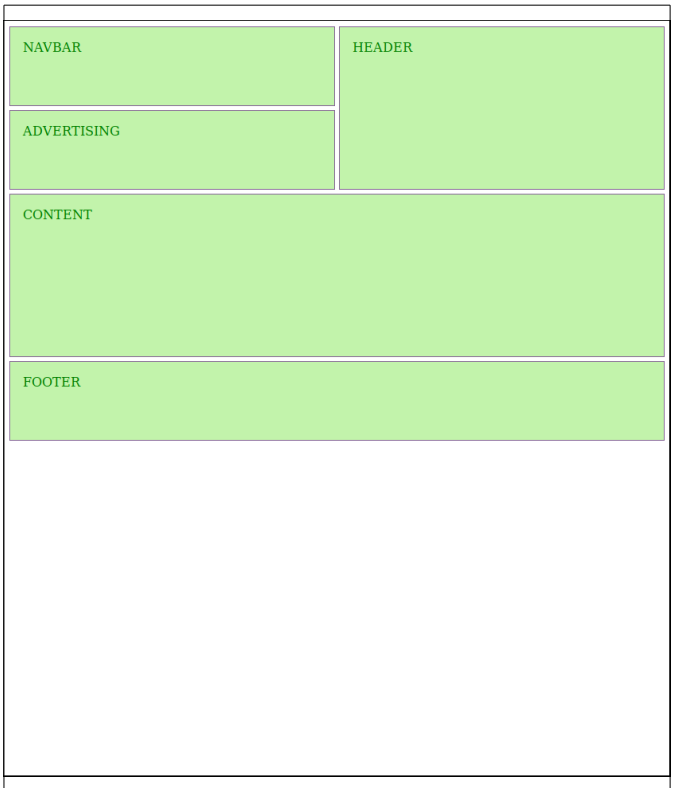
style.css

```
* {box-sizing: border-box;}

.wrapper {
	max-width: 1400px;
	margin: 0 auto;
}

.wrapper > div {
	border: 1px solid rgb(133,100,150);
	background-color: rgba(133,231,88,.5);
	padding: 1em;
	color: green;
}

.header {
	grid-area: header;
}
.navbar {
	grid-area: navbar;
}
.content {
	grid-area: content;
}
.footer {
	grid-area: footer;
}
.ads {
	grid-area: ads;
}
```

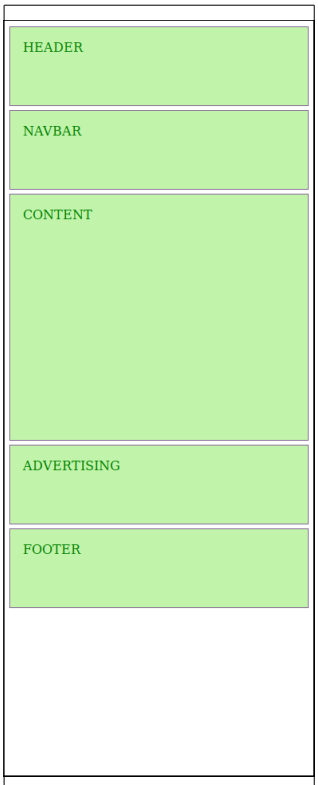
```
.wrapper {
	display: grid;
	grid-gap: 5px;
	grid-template-columns: 1fr;
	grid-auto-rows: minmax(100px, auto);
	grid-template-areas:
		"header"
		"navbar"
		"content"
		"content"
		"content"
		"ads"
		"footer";
}

@media only screen and (min-width: 600px) {
	.wrapper {
		grid-template-columns: repeat(2, 1fr);
		grid-template-areas:
			"navbar header"
			"ads header"
			"content content"
			"content content"
			"footer footer";
	}
}

@media only screen and (min-width: 1200px) {
	.wrapper {
		grid-template-columns: repeat(3, 1fr);
		grid-template-areas:
			"header header header"
			"navbar navbar navbar"
			"content content ads"
			"content content ads"
			"content content ..."
			"footer footer footer";
	}
}
```

Rezultat:

Zaslon širine veće od 1200px



Zaslon širine između 600px i 1200px



Zaslon širine manje od 600px

ANALIZA *CSS grid layout* metode

PREDNOSTI:

- jednostavno korištenje
- minimalan kod
- proizvoljno pozicioniranje elemenata bez obzira na redoslijed unutar html dokumenta
- jednostavno određivanje širine elementa flexbox metodom
- jednostavno određivanje razmaka između elemenata
- *grid-template-areas* svojstvo – jednostavno određivanje veličine i širine pojedinog elementa

MANE:

- podrška preglednika nije potpuna
- još uvijek nije pod W3C standardom

Zaključak:

CSS grid layout ili samo *CSS grid* je nova tehnika koja omogućuje dizajnerima kreiranje kompleksnih responzivnih web rješenja na jednostavan način uz minimalnu količinu koda. Iako još uvijek nije u velikoj upotrebi, njegovo vrijeme tek dolazi što pokazuje i sve veća zainteresiranost dizajnera za tu metodu. Treba napomenuti da je veliki “vjetar u leđa” ovoj metodi dala i Mozilla koja je unutar nove verzije svoga popularnog web preglednika “Firefox” dodala *grid ispector* alat koji olakšava izradu responzivnih stranica pomoću CSS grida.

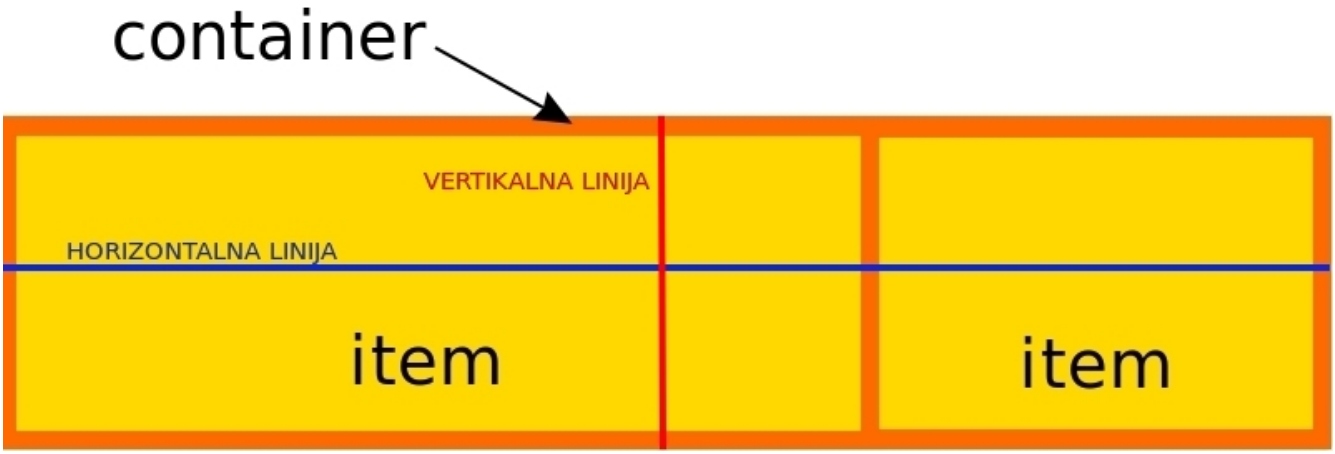
Flexbox

Modul Flexbox (fleksibilni okvir) ima za cilj pružiti učinkovitiji način postavljanja, poravnavanja i distribucije prostora između stavki u spremniku (*container*) čak i kada je njihova veličina nepoznata i / ili dinamična.

Glavna ideja Flexboxa je da *container* mijenja širinu, visinu i redoslijed svojih stavki kako bi najbolje ispunio raspoloživi prostor. *Flex container* proširuje elemente tako da popuni sav slobodni prostor ili ih smanjuje kako bi spriječio *overflow (prelijevanje elemenata)*.

Svojstva Flexboxa

Svojstva Flexboxa možemo podijeliti u dvije skupine – *Flex Container* i *Flex Items.*



Flex Container

- *html* blok unutar kojeg se nalaze elementi (Flex Items) *flexboxa*

Prije svega da bi mogli koristiti Flexbox modul moramo postaviti svojstvo **display: flex;**

```
.container{
     display: flex;
}
```

Direction

Ukoliko želimo promijeniti smjer rasporeda elemenata unutar c*ontainera, koristimo svojstvo* ***flex-direction.***

- ***row;*** → osnovni zadani raspored elemenata po horizontalnoj osi
- ***row-reverse***;→ obrnuti raspored elemenata po horizontalnoj osi
- ***column;*** → osnovni raspored elemenata po vertikalnoj osi
- ***column-reverse;*** → obrnuti raspored elemenata po vertikalnoj osi

```
.container{
    flex-direction: row | row-reverse | column | column-reverse;
}
```

Wrap

Flex-wrap svojstvo omogućava nam da odredimo na koji način se elementi “prelijevaju” unutar *containera.*

- ***nowrap;*** → bez omota, elementi ostaju u jednom redu
- ***wrap;*** → elementi se “prelijevaju” odozgo prema dolje
- ***wrap-reverse;*** → elementi se “prelijevaju” odozdo prema gore

```
.container{
    flex-wrap: nowrap | wrap | wrap-reverse;
}
```

Flow

Flex-flow je svojstvo kojim skračeno definiramo ***flex-direction*** i ***flex-wrap***.

```
.container{

     flex-flow: <flex-direction> <flex-wrap>;
     /* osnovno je row nowrap */
}
```

Justify Content

Ovime određujemo poravnjanje i poziciju elemenata unutar *containera* po horizontalnoj osi.

- ***flex-start;*** → elementi plutaju (*float*) na lijevo
- ***flex-end;*** → elementi plutaju na desno
- ***center;*** → elementi su u sredini *containera*
- ***space-between;*** → razmak između elemenata je jednak dok su prvi i zadnji element na rubovima
- ***space-around;*** → razmak između elemenata je jednak ali i različit od razmaka na rubovima
- ***space-evenly;*** → razmak između elemenata i na rubovima je jednak

```
.container{

     justify-content: flex-start | flex-end | center | space-
     between | space-around | space-evenly;
}
```

Align Items

Ovime određujemo kako su elementi posloženi po vertikalnoj osi.

- ***flex-start;*** → svi elementi plutaju prema gore
- ***flex-end;*** → svi elementi plutaju prema dolje
- ***center;*** → elementi su centrirani po vertikalnoj osi *containera*
- ***stretch;*** → elementi su vertikalno razvučeni od početka do kraja *containera* (osnovno)
- ***baseline;*** → elementi *containera* su poravnati po glavnoj horizontalnoj osi

```
.container{

    align-items: flex-start | flex-end | center | baseline |
    stretch;

}
```

Align Content

Ako postoji višak prostora na vertikalnoj osi *containera*, ovo svojstvo određuje na koji način je prostor raspoređen unutar *containera*.

- ***flex-start;*** → elementi su pozicionirani na početak *containera* i prirodno se “prelijevaju” u novi red
- ***flex-end;*** → elementi su pozicionirani na kraj *containera* i prirodno se “prelijevaju” u novi red
- ***center;*** → elementi su pozicionirani na sredinu *containera*, a višak prostora se nalazi na početku i kraju *containera*
- ***space-between;*** → višak prostora je raspoređen između redova *containera*
- ***space-around;*** → višak prostora je podjednako raspoređen na početku i kraju *containera* te između redova
- ***stretch;*** → redovi *containera* su razvučeni tako da ispune sav raspoloživi prostor

```
.container{

     align-content: flex-start | flex-end | center | baseline |
     stretch;

}
```

Flex Item

- elementi unutar Flex Containera koji imaju zasebna css svojstva

Order

Kao što i sam naziv svojstva govori, ovime određujemo raspored elemenata na način da odredimo redni broj elementa u odnosu na sve druge elemente *containera.*

```
.item{

     order: <integer>; /* osnovno je 0 */

}
```

Flex Grow

Svojstvo kojim određujemo koliko puta je neki element veći u odnosu na većinu drugih elementa.

Ako je ***flex-grow*** svih elemenata jednak, elementi će imati jednaku širinu unutar *containera*.

```
.item{

     flex-grow: <broj>; /* osnovno je 0 */

}
```

Flex Shrink

Svojstvo elementa da se smanji ako je potrebno. Brojem određujemo koliko puta je element manji od ostalih elemenata.

```
.item{
     flex-shrink: <broj>; /* osnovno je 1 */
}
```

Flex Basis

Ovime definiramo širinu elementa prije nego što je prostor unutar *containera* raspoređen. Širina može biti određena u postocima, pixelima, rem-ovima ili bilo kojoj drugoj jedinici koja određuje veličinu u css-u.

```
.item{
     flex-basis: <širina> | auto; /* osnovno je “auto” */
}
```

Flex

Globalno svojstvo za određivanje flex-grow, flex-shrink i flex-basis.

```
.item{
     flex: none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-
     basis'> ]
}
```

Align Self

Ovo svojstvo nam dopušta da odredimo posebno poravnanje za pojedini element koje se razlikuje od globalnog poravnanja ***align-items.***

```
.item{

     align-self: auto | flex-start | flex-end | center | baseline |
     stretch;
}
```

ANALIZA Flexbox metode

PREDNOSTI:

- puna podrška preglednika
- slobodni prostor unutar *containera* je u potpunosti ispunjen
- responzivan dizajn bez potrebe korištenja *@media* svojstva
- minimalan kod i jednostavnost primjene
- svakom elementu unutar *containera* se mogu promijeniti svojstva bez utjecaja na druge elemente
- ukomponiran u Bootstrap i druge css *frameworke*

MANE:

- nije pogodan za veće projekte

Zaključak:

Kao što i samo ime govori (flex), Flexbox nam nudi fleksibilno rješenje za izradu responzivnog dizajna web stranica čak i bez potrebe korištenja ***@media*** svojstva css-a što je velika prednost jer smanjuje vrijeme izrade, no upravo zbog te fleksibilnosti nije pogodan za veće projekte gdje svaki element treba biti na određenom mjestu bez promjene pozicije smanjivanjem širine zaslona. Flexbox svojstvo se vrlo često koristi za izradu navigacijskih rješenja upravo zbog toga što se elementi (u ovom slučaju poveznice stranice) dinamički prilagođavaju širini zaslona.

ZAKLJUČAK

Danas imamo mnogo načina za izradu responzivnih web stranica i sve načine možemo međusobno kombinirati što nam daje maksimanu fleksibilnost izrade svih vrsta dizajna te za sve širine uređaja.

Najbrži način izrade responzivnog dizajna je kroz već gotova rješenja unutar css *frameworka* kao što je Bootstrap grid i flexbox, no ako želimo responzivnost na maksimalnoj mogućoj razini bez viška koda, najbolji izbor je izrada *custom* rješenja gore navedenim tehnikama.