Een eenvoudige pagina layout in HTML / CSS maken met een *Grid* system

## inleiding

In deze opdracht maken we een eenvoudige webpagina maken waarbij we gebruik maken van een CSS Grid ook wel flexgrid genoemd. Met behulp van het CSS grid is het in HTML / CSS mogelijk om je elementen volledig flexibel over je webpagina te verdelen. In tegenstelling tot flexbox is een Grid syteem meer gericht op het indelen van de layout. Flexbox kan dit ook echter met wat beperkingen. Met het grid systeem bestaan deze beperkingen niet of in ieder geval veel minder. Kort gezegd zou je kunnen zeggen dat een gridsysteem gebruikt wordt voor de layout terwijl je met de flexbox meer individuele onderdelen kunt vormgeven. Het is dus zeer goed mogelijk om een flexbox binnen een CSS grid systeem te gebruiken. Andersom kan ook: een grid systeem in een flexbox. Dat laatste zul je overigens wat minder toepassen.

Het gridsysteem is meer uitgebreid dan de flexbox, wat betekend dat het ook iets complexer is om toe te passen. In deze tutorial houden we het eenvoudig en zul je zien dat het best meevalt

## Voorkennis:

* Basis kennis HTML
* Basis kennis CSS
* Kennis CSS selectors
* Basis kennis flexbox
* Basiskennis IDE programmeeromgeving

## Leerdoelen

* Je kunt op middels het HTML / CSS gridsysteem een responsive weblayout maken
* Je kunt verschillende grid systeem eigenschappen toepassen om een gridlayout naar eigen wens te bouwen
* Je kent het verschil tussen de flexbox en flexgrid en je weet welke methode je wanneer het beste toe kunt passen.

## De basis

We starten met een basis project. Maak deze aan in een aparte subfolder van je webroot locatie op je PC. Open je favoriete IDE om deze folder te openen en maak een nieuw *index.html* bestand. Maak tevens in een aparte subfolder die je *css* noemt een extern stylesheet bestand. Noem deze *styles.css*

Je structuur ziet er nu als volgt uit:

webroot

│ index.html

└───css

styles.css

Nu hebben we de basis van je project gelegd.

## De indexpagina

Open index.html en maak een standaard HTML pagina met de standaard elementen erin. Vraag je docent indien je niet weet hoe dit gaat.

Maak een link naar de het externe stylesheet bestand. Zet deze verwijzing in de <head> van je html document.

<link rel="stylesheet" href="css/styles.css">

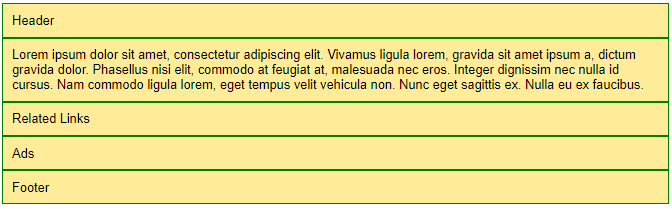
Maak nu de volgende indeling in je <body> van je *index.html* document:

<main>  
 <header class="main-header">  
 Header  
 </header>  
 <article class="main-content">  
 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vivamus ligula lorem, gravida sit amet ipsum a, dictum gravida dolor. Phasellus nisi elit, commodo at feugiat at, malesuada nec eros. Integer dignissim nec nulla id cursus. Nam commodo ligula lorem, eget tempus velit vehicula non. Nunc eget sagittis ex. Nulla eu ex faucibus.  
 </article>  
 <aside class="related-links">  
 Related Links  
 </aside>  
 <aside class="ads">  
 Ads  
 </aside>  
 <footer class="main-footer">  
 Footer  
 </footer>  
</main>

Open het stylesheet bestand styles.css en voeg de volgende styling toe:

html {  
 box-sizing: border-box;  
 font-family: arial, helvetica, sans-serif;  
}  
img {  
 max-width: 100%;  
}  
\*,\*:before,\*:after {  
 box-sizing: inherit;  
}  
  
main {  
 margin: 30px;  
}  
  
main > \* {  
 border: 2px solid green;  
 background-color: #ffec99;  
 padding: 10px;  
}

Bekijk nu de webpagina in je browser:



Zoals je ziet niets bijzonders, alles staat onder elkaar…

Geef aan waarom dit zo is?

Antwoord:

## Meer CSS styling toevoegen

Om nu gebruik te maken van het grid systeem moeten we net als bij de flexbox een container aanwijzen. We gebruiken hier de <main> element als het *container* element. Geef dit element de volgende styling in je stylesheet bestand:

main {  
 margin: 30px;  
 display: grid;  
 grid-gap: 5px;  
 grid-template-areas:  
 "header"  
 "content"  
 "related"  
 "ads"  
 "footer";  
}

Wat je nu ziet is dat dit element wordt ingesteld als grid systeem middels de display eigenschap:

display: grid;

Dit maakt van deze container nu een CSS grid systeem met alle eigenschappen die daarbij horen. Een daarvan is *de grid-gap*. Zoals het woord al zegt: dit maakt een gap (lees: gat of ruimte) tussen de onderliggende elementen in pixels. Doe je dit niet dan liggen alle child elementen tegen elkaar aan.

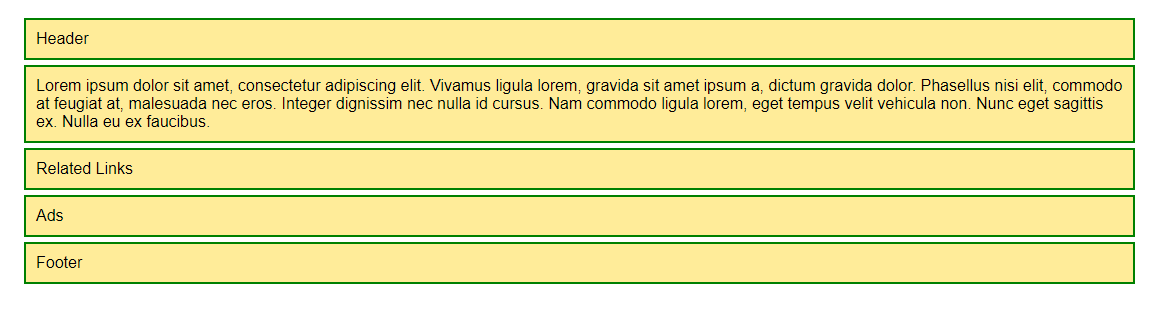
Verder is hier interresant de eigenschap: grid-template-areas:

grid-template-areas:  
 "header"  
 "content"  
 "related"  
 "ads"  
 "footer";

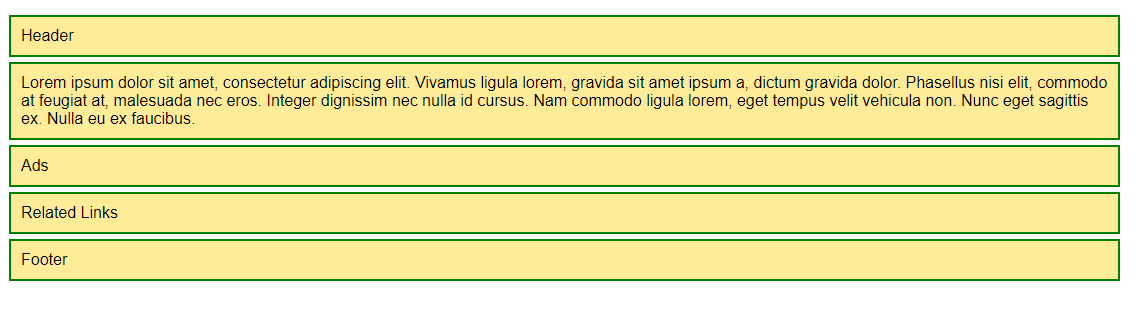
Om deze eigenschap te laten werken moeten we de child elementen binnen het grid als het ware een *label* geven. Dat doen we met *grid-area* eigenschap zoals de volgende CSS code die je *toevoegt* aan je styles.css bestand laat zien:

.main-header {  
 grid-area: header;  
}  
  
.main-content {  
 grid-area: content;  
}  
  
.related-links {  
 grid-area: related;  
}  
  
.ads {  
 grid-area: ads;  
}  
  
.main-footer {  
 grid-area: footer;  
}

Bekijk de webpagina nu nog maar eens en zie hoe de “gap” tussen de elementen eruit ziet:



Zou je de volgorde in de *grid-template-areas* wijzigen (wissel maar eens de *related* en *ads* om) dan heeft dat effect op de layout volgorde in je browser:  
  
main {  
 margin: 30px;  
 display: grid;  
 grid-gap: 5px;  
 grid-template-areas:  
 "header"  
 "content"  
 **"ads"  
 "related"** "footer";  
}



## Een mediaquery toevoegen

Wanneer we de browser openen in een mobiele telefoon zou de laatste layout opzich bruikbaar kunnen zijn. Alle elementen onder elkaar is voor een smal display de meest toegepaste layout. Op een desktop computer is dit wat minder gebruikelijk. We gaan middels een media query detecteren wat de huidige schermbreedte is. Wanneer deze boven de 500 pixels uitkomst passen we een andere layout toe. We doen dat als volgt:

@media (min-width: 500px) {  
 main {  
 grid-template-columns: repeat(3,1fr);  
 grid-template-areas:  
 "header header header"  
 "content content related"  
 "content content ads"  
 "footer footer footer";  
 }  
}  
  
@media (min-width: 800px) {  
 main {  
 grid-template-columns: repeat(4,1fr);  
 grid-template-areas:  
 "header header header header"  
 "related content content ads"  
 "footer footer footer footer";  
 }  
}

Voeg bovenstaande CSS code toe aan styles.css en zie de verschillende *breakpoints (lees: punten waarbij op basis van verschillende breedtes bepaalde CSS onderdelen worden uitgevoerd).* We gebruiken hier feitelijk 2 breakpoints gemaakt door de twee mediaqueries (minimaal 500 pixels en 800 pixels breed). Is geen van beide waar (dus kleiner dan 500 pixels) dan kunnen we ervan uitgaan dat we hebben te maken met een mobiel apparaat zoals bijvoorbeeld een telefoon. Omdat deze CSS code zich niet binnen een mediaquery bevind en dus de standaard CSS betreft spreken we hier over een *mobile-first design*. Oftwel de CSS code voor een mobiel apparaat van kleiner dan 500 pixels breed zal altijd worden geladen, de andere pas wanneer de breedte over de 500 of 800 pixels gaat.

Let ook vooral op hoe de indeling van de grid-template-areas wordt aangegeven in de CSS

In geval van een minimale breedte van 800 pixels (dekstop computer) bestaat de *header* uit 4 child elementen naast elkaar, daaronder van links naar rechts: 1 x related, 2 x content en 1 x ads. Vervolgens daaronder de footer in 4 elementen.

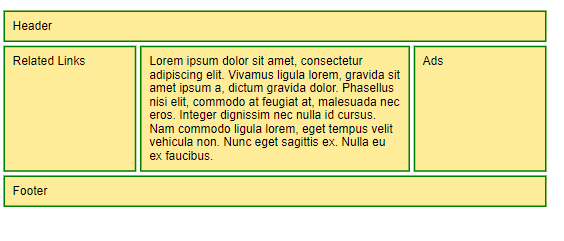
grid-template-areas:  
 "header header header header"  
 "related content content ads"  
 "footer footer footer footer";  
  
Om dit correct te laten werken is het noodzakelijk aan te geven met welke kolommen we hebben te maken. Dit wordt gedaan door de grid-template-columns eigenschap. We hebben hier te maken met 4 kolommen en in feite wordt het volgende uitgevoerd:

grid-template-columns: 1fr 1fr 1fr 1fr;

Dit is korter geschreven middels de repeat functie:

grid-template-columns: repeat(4,1fr);

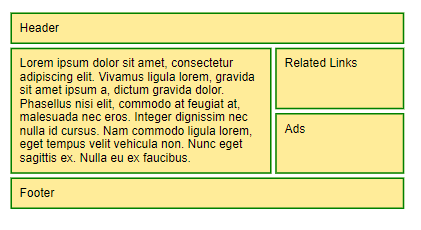
Dit betekend dat de totale breede wordt verdeeld over vier *fractional* units (1fr).

  
Zo wordt het bij het 500 pixels breakpoint de indeling weer iets anders. Hier worden 3 kolommen gebruikt. Denk bijvoorbeeld aan een iPad of andere tablet.

grid-template-areas:  
 "header header header"  
 "content content related"  
 "content content ads"  
 "footer footer footer";  
 }

De kolommen worden dan dus als volgt ingesteld:

grid-template-columns: repeat(3,1fr);



## Schermhoogte instellen

Laten we nu nog wat meer gebruik maken van de schermhoogte. Wijzig de CSS code tot het volgende:  
  
html {  
 box-sizing: border-box;  
 font-family: arial, helvetica, sans-serif;  
}  
  
body {  
 margin: 0;  
}  
  
img {  
 max-width: 100%;  
}  
\*,\*:before,\*:after {  
 box-sizing: inherit;  
}  
  
main {  
 margin: 0px;  
 display: grid;  
 grid-gap: 5px;  
 grid-template-areas:  
 "header"  
 "content"  
 "related"  
 "ads"  
 "footer";  
}  
  
main > \* {  
 border: 2px solid green;  
 background-color: #ffec99;  
 padding: 5px;  
}  
  
.main-header {  
 grid-area: header;  
 height: 10vh;  
}  
  
.main-content {  
 grid-area: content;  
 height: 60vh;  
  
}  
  
.related-links {  
 grid-area: related;  
 height: 10vh;  
}  
  
.ads {  
 grid-area: ads;  
 height: 10vh;  
}  
  
.main-footer {  
 grid-area: footer;  
 height: 10vh;  
}  
  
@media (min-width: 500px) {  
 html {  
 overflow: hidden;  
 }  
  
 main {  
 grid-template-columns: repeat(3,1fr);  
 grid-template-areas:  
 "header header header"  
 "content content related"  
 "content content ads"  
 "footer footer footer";  
 }  
}  
  
@media (min-width: 800px) {  
 html {  
 overflow: hidden;  
 }  
 main {  
 grid-template-columns: repeat(4,1fr);  
 grid-template-areas:  
 "header header header header"  
 "related content content ads"  
 "footer footer footer footer";  
 }  
}

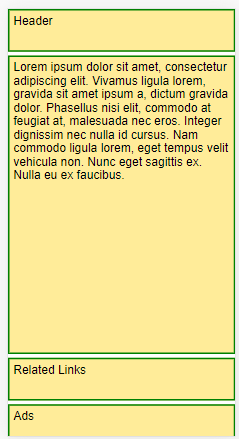
Er zijn een paar wijzigingen ten opzichte van wat je reeds had toegevoegd. Als eerst is de marge van de body en het main element even op 0 gezet. Uiteraard mag je dit laten staan maar dan moet je rekening houden met de totale hoogte van je layout. Die wordt dan wat anders.

Ook is de marge van de body op 0 gezet:  
  
  
body {  
 margin: 0;  
}

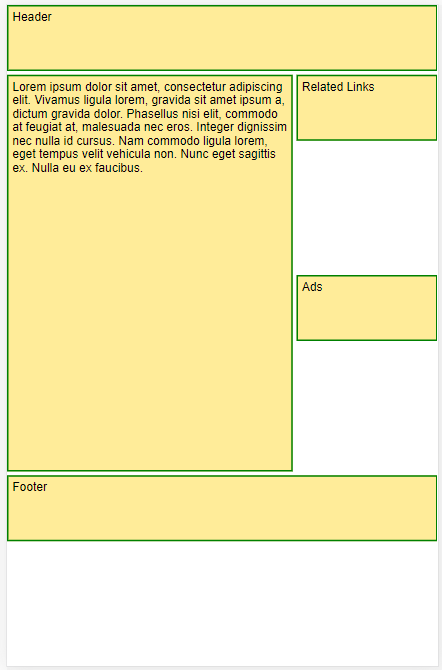
Verder is gebruik gemaakt van de vh (view-height) unit om de hoogte van de header, content, related, adds en footer in te stellen (height: 10vh / 60vh / 10vh / 10vh / 10vh).

Verder is in de media-query gezorg dat de scrollbars uitgezet worden indien deze worden uitgevoerd:

html {  
 overflow: hidden;  
}

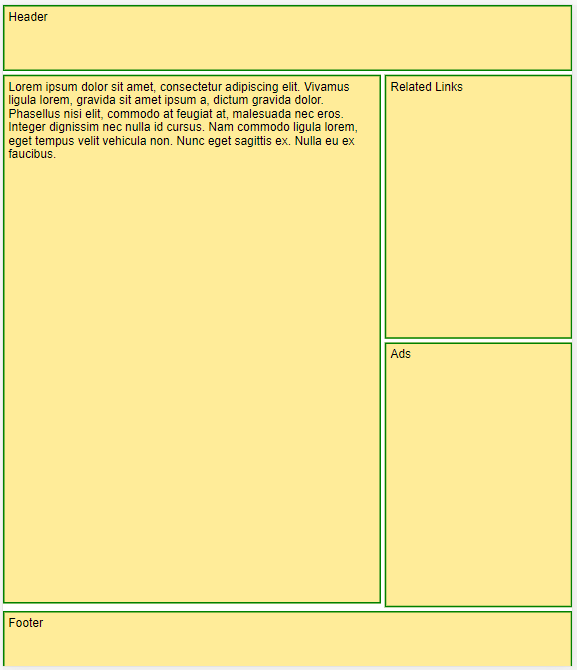
Bekijk je nu de pagina in mobiele weergave dan ziet hij er prima uit:  
  


Doe je dit echter in **tablet** weergave van minimaal 500 pixel dan gaat het niet helemaal lekker:

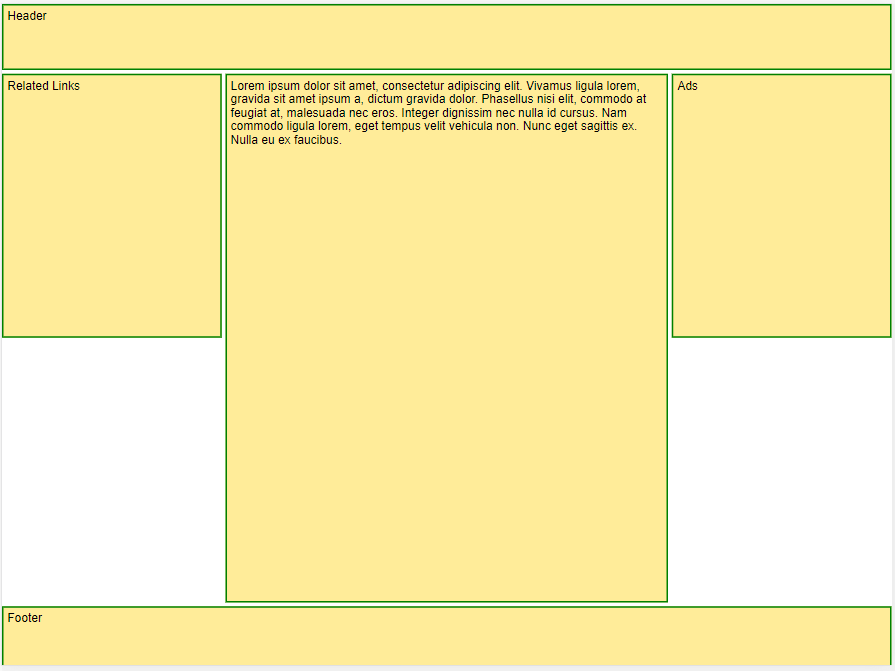


Zoals je ziet zijn is de content, related links, adds niet hoog genoeg. We hebben het feitelijk over 3 rijen die in totaal een 100vh zouden moeten hebben. Laten we dit aanpassen in de CSS van de juiste mediaquery:

@media (min-width: 500px) {  
 html {  
 overflow: hidden;  
 }  
  
 main {  
 grid-template-columns: repeat(3,1fr);  
 grid-template-areas:  
 "header header header"  
 "content content related"  
 "content content ads"  
 "footer footer footer";  
 }  
 .related-links, .ads {  
 height: 40vh;  
 }  
 .main-content {  
 height: 80vh;  
 }  
}



Hetzelfde prolbeem treed op bij de dekstop variant:



Laten we deze aanpassing ook doorvoeren, maar dan in de 800 pixel breakpoint mediaquery:

@media (min-width: 800px) {  
 html {  
 overflow: hidden;  
 }  
 main {  
 grid-template-columns: repeat(4,1fr);  
 grid-template-areas:  
 "header header header header"  
 "related content content ads"  
 "footer footer footer footer";  
 }  
  
 .related-links, .ads {  
 height: 80vh;  
 }  
}

