Web technology

Les 5: CSS - Grid Layout

Kristof Michiels

In deze presentatie

- Grid Layout?
- Basisgebruik
- Ruimte tussen de elementen met gap
- Positioneren op de grid
- Werken met *named areas*
- *Grid-template-rows*
- Grid-elementen uitlijnen
- Strategie voor grid-gebruik

CSS Grid Layout?

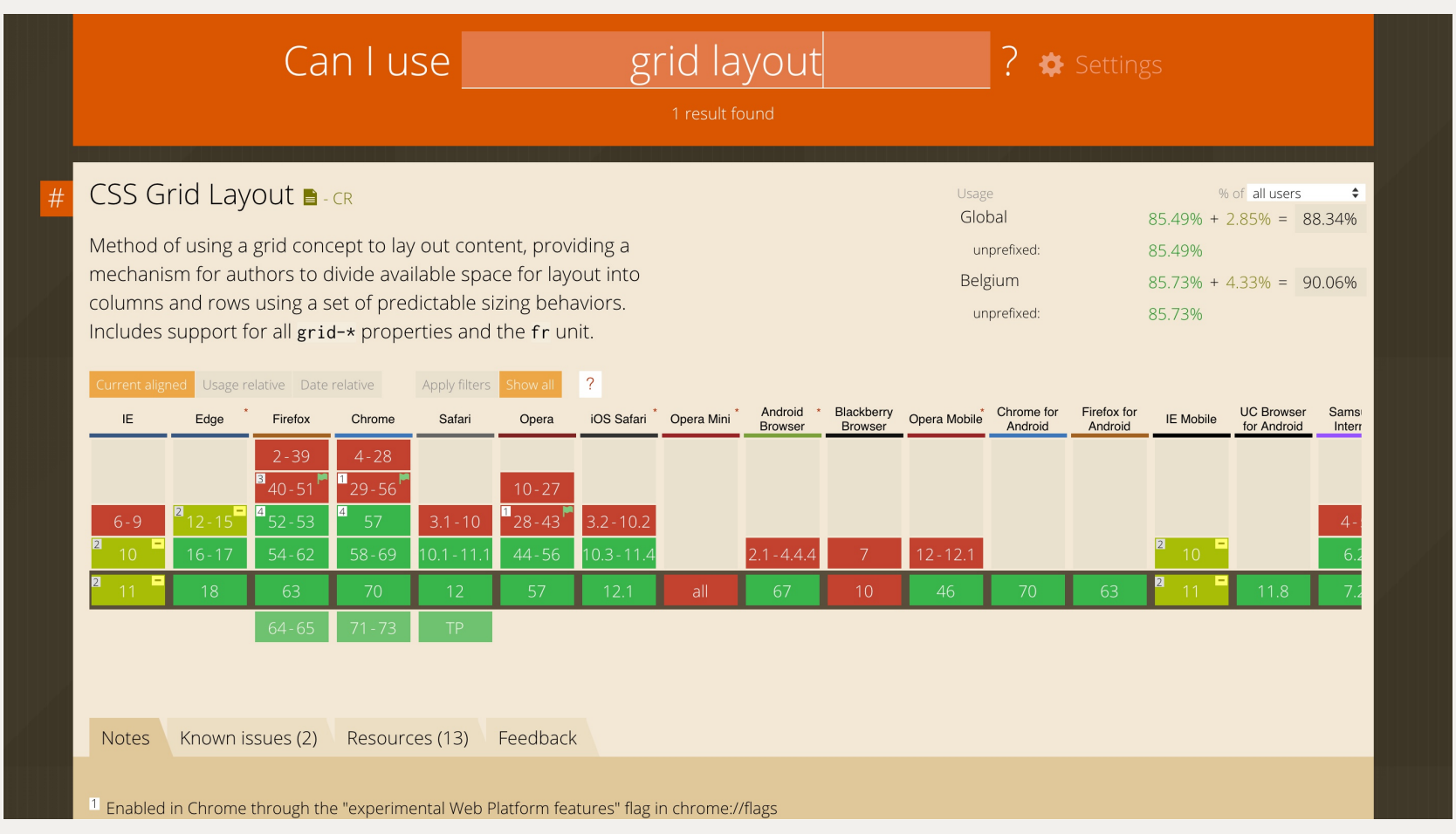
CSS Grid Layout

- Grid layout is dé CSS specificatie voor layouts
- Eerste specificatie die gericht ontwikkeld is voor pagina layout
- Grid Layout doet exact wat de naam suggereert: je maakt er rooster-layouts mee in CSS :-)
- Het gaat hier om 2-dimensionele layouts: een raster dat bestaat uit rijen en kolommen
- Voorheen gebruikten we floats en positioning
- Deze technieken (met name positioning) zijn nog bruikbaar in specifiekere omstandigheden

Grid Layout relatief recent

- Officieel bruikbaar sinds april 2017
- Momenteel ondersteuning bij >96% van de webgebruikers wereldwijd
- We weten dit via de site http://caniuse.com

Can i use Grid Layout?



https://caniuse.com/

Grid Layout basisvoorbeeld

Grid Layout basisvoorbeeld

- Een ul-element met daarin zes li-elementen
- Het element waarvan je een grid wenst te maken noemen we de grid container
- Deze ul zal in dit voorbeeld onze grid container worden

```
<ul>
    <li>1</li>
    <li>2</li>
    <li>3</li>
    <li>4</li>
    <li>5</li>
    <li>6</li>
</ul>
```

Grid Layout basisvoorbeeld

Je maakt er een container van binnen CSS met *display: grid;*

```
ul {
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;
    gap: 3rem;
    height: 100vh;
    padding: 3rem;}

li {
    padding: 3rem;
    color: white;
    font-size: 3rem;
    text-align: center;
    padding-top: 8rem;}
```

Grid Layout basisvoorbeeld

- Uiteraard niet vergeten er een achtergrondkleurtje op te zetten, anders zie je niet veel ;-)
- Merk ook op dat we hier gebruik maken van de :nth-child() selector: handig!

```
li:nth-child(1) {background-color: firebrick;}
li:nth-child(2) {background-color: yellowgreen;}
li:nth-child(3) {background-color: pink;}
li:nth-child(4) {background-color: deepskyblue;}
li:nth-child(5) {background-color: yellow;}
li:nth-child(6) {background-color: plum;}
```

Grid Layout basisvoorbeeld

- We verdelen onze grid in ons voorbeeld in drie kolommen: de browser zorgt voor "grid-cellen"
- We doen dit met de *grid-template-columns* eigenschap
- We geven als waardes voor elk van de kolommen die we willen, gescheiden door een spatie, hun breedte
- Deze breedte kan met rem, %, vw-vh, px, ... worden weergegeven
- Er is voor Grid Layout ook een nieuwe flexibele lengte-eenheid gecreëerd: de fr
- fr staat voor een “fraction” van de beschikbare ruimte in de grid container
- Met 3x 1fr wordt in ons voorbeeld de beschikbare ruimte evenredig gedeeld door 3

Conclusie basisvoorbeeld

- Eerst: bepalen wie de grid container wordt (hier onze ul)
- De kind-elementen worden positioneerbaar binnen deze container (hier onze li's)
- Je deelt de container op in één of meerdere kolommen
- Je zet bij voorkeur afmetingen op de grid container en niet op de grid elementen
- De browser plaatst de kinderen automatisch op de grid
- We noemen dit de "impliciete grid" of de "natuurlijke flow": de browser bepaalt ook zelf hoeveel rijen nodig zijn in onze grid
  - We kunnen daar zelf indien nodig op ingrijpen met de *grid-template-rows* eigenschap (zie verder)

Ruimte tussen de elementen met gap

De gap eigenschap

- Je zag het me al gebruiken in het basisvoorbeeld
- We creëren ruimte tussen de elementen met gap
- Deze eigenschap is een verkorte notatie van column-gap en row-gap
- Deze twee laatsten kunnen ook afzonderlijk worden gebruikt

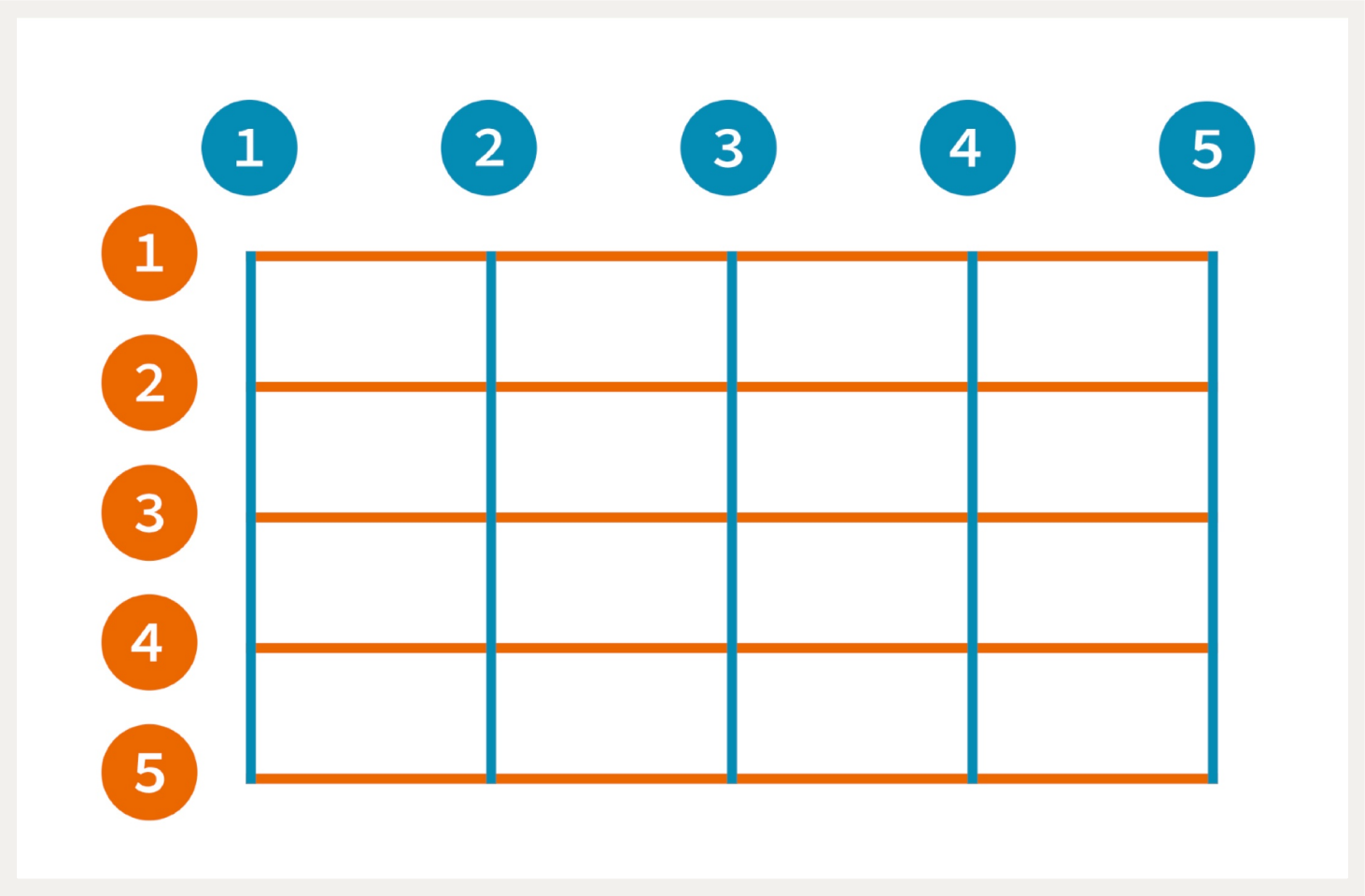
```
ul {
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;
    column-gap: 3rem;
    row-gap: 1rem;
    border: .1rem solid green;
}
```

Positioneren op de grid

Positioneren op de grid

- We zagen in het basisvoorbeeld de werking van de "natuurlijke flow" van elementen op de grid
- We kunnen grid-elementen ook gericht gaan positioneren
- Wat we hier dan doen vanuit CSS is ingrijpen in de volgorde die we in de html aan onze elementen hebben gegeven
- We kunnen hiervoor gebruik maken van gridlijn-nummers waaruit elke grid bestaat

Positioneren op de grid: gridlijn-nummers



Bvb. vier kolommen geeft 5 grid-lijnen, vier rijen geeft 5 grid-lijnen

Voorbeeld van positioneren

- We vertrekken terug van ons basisvoorbeeld
- Door de eigenschappen grid-row en grid-column toe te passen op de grid-elementen grijpen we in op de "natuurlijke flow"
- Merk ook op dat de overige elementen automatisch een nieuwe plaats krijgen op de grid
- Je kan met deze techniek ingrijpen op zoveel elementen als nodig is

```
li:nth-child(6) {
    grid-column: 1 / 2;
    grid-row: 1 / 2;
}
```

Positioneren op de grid

- We kunnen zelfs witruimte creëren op onze grid

```
li:nth-child(5) {
    grid-column: 1 / 2;
    grid-row: 3 / 4;
}
```

Positioneren op de grid

- En we kunnen elementen uitsmeren over meerdere grid-cellen

```
li:nth-child(2) {
    grid-column: 2 / 4;
    grid-row: 1 / 2;
}

li:nth-child(3) {
    grid-column: 2 / 3;
    grid-row: 2 / 4;
}

li:nth-child(4) {
    grid-column: 3 / 4;
    grid-row: 2 / 4;
}
```

Werken met *named areas*

Werken met *named areas*

- Een alternatieve manier voor het positioneren van elementen op een grid is het gebruiken van *named areas*
- Je hoeft dit dus niet te gebruiken, maar kan wel handig zijn voor complexe grids
- Je geeft elk grid-element een naam met de grid-area eigenschap. We gebruikten in ons voorbeeld de letters a t.e.m. e.
- Vervolgens voeg je de grid-template-areas eigenschap toe aan de grid container
- Daarin beschrijf je de grid met behulp van de gekozen namen

Named-areas: basisvoorbeeld

- Eerst geef je alle grid-elementen een naam met de grid-area -eigenschap

```
li:nth-child(1) {grid-area: a; background-color: firebrick;}
li:nth-child(2) {grid-area: b; background-color: yellowgreen;}
li:nth-child(3) {grid-area: c; background-color: pink;}
li:nth-child(4) {grid-area: d; background-color: deepskyblue;}
li:nth-child(5) {grid-area: e; background-color: yellow;}
li:nth-child(6) {grid-area: f; background-color: plum;}
```

Named-areas: basisvoorbeeld

- Vervolgens beschrijf je in de grid container met grid-template-areas de grid
- Je maakt bij deze beschrijving gebruik van de gekozen grid-area -namen

```
ul {
    display: grid;
    grid-template-columns: 1fr 1fr 1fr;
    gap: 3rem;
    height: 100vh;
    padding: 3rem;
    grid-template-areas:
        "f    e    d"
        "c    b    a";
}
```

Named-areas: basisvoorbeeld

- Om de elementen te spreiden over verschillende cellen herhalen we de naam van het element
- De gebieden die we met deze manier creëeren moeten rechthoekig zijn
- Om witruimte te laten (een lege cel), gebruik je een punt

```
ul {
    display: grid;
    grid-template-columns: 1fr 1fr 1fr 1fr;
    gap: 3rem;
    height: 100vh;
    padding: 3rem;
    grid-template-areas:
        "f    e    e    d"
        "c    b    .    a";
}
```

Grid-template-rows

Grid-template-rows

- Net zoals grid-template-columns kan je ook grid-template-rows gaan beschrijven
- Niet verplicht: gebruik enkel indien een goede reden!
- Ook hier kan je de verschillende meeteenheden gebruiken (ook fr)
- https://developer.mozilla.org/en-US/docs/Web/CSS/grid-template-rows

```
.elementen {
    display: grid;
    grid-template-columns: 1fr 1fr 1fr 1fr;
    grid-template-rows: 25rem 25rem 25rem;
    gap: 2rem; }
```

De repeat()-functie

- Indien je dat wil kan je voor de eigenschappen grid-template-columns en grid-template-rows gebruik maken van de repeat()-functie
- Hiermee kan je de notatie verkorten
- De code hieronder gebruikt de repeat()-functie om hetzelfde te schrijven als op de vorige slide
- Meer info: https://developer.mozilla.org/enUS/docs/Web/CSS/repeat

```
.elementen {
    display: grid;
    grid-template-columns: repeat(4,1fr); /* kan ook: 0.5fr repeat(2,1fr) 0.75fr */
    grid-template-rows: repeat(3,25rem);
    grid-gap: 2rem; }
```

Grid-elementen uitlijnen

Grid-elementen uitlijnen op de kolommen-as

- We kunnen de elementen binnen een grid op verschillende manieren gaan uitlijnen op de kolommen-as
- Om uit te lijnen gebruiken we de eigenschap align-items op de grid-container
- De standaardwaarde voor deze eigenschap is stretch (element neemt alle hoogte in)
- start: neem enkel de natuurlijke hoogte in en plaats jezelf bovenaan
- Andere waarden: end, center, ...
- We kunnen de align-self eigenschap gebruiken om voor individuele elementen uit te lijnen
- https://developer.mozilla.org/en-US/docs/Web/CSS/align-items

Grid-elementen uitlijnen op de rijen-as

- We kunnen de elementen binnen een grid ook gaan uitlijnen op de rijen-as
- Om uit te lijnen gebruiken we de eigenschap justify-items
- De standaardwaarde voor deze eigenschap is stretch (element neemt alle breedte in)
- start: neem enkel de natuurlijke breedte in en plaats jezelf links
- Andere waarden: end, center, ...
- We kunnen de justify-self eigenschap gebruiken om voor individuele elementen uit te lijnen
- https://developer.mozilla.org/en-US/docs/Web/CSS/justify-items

Grid strategie

Grid strategie

- Je kan zonder problemen een layout maken waarin meerdere grids gebruikt worden.
- Deze grids kunnen naast elkaar bestaan
- Maar het is ook mogelijk dat een grid element van één grid tevens ook tot een grid container wordt uitgebouwd
- We spreken dan over geneste grids

Grid strategie

- Wanneer je een pagina gaat vormgeven, dan ga je op basis van je ideeën en (eenvoudige) schetsen bepalen waar je één of meerdere grids wil gaan gebruiken
- Het is zeker niet de bedoeling om van je pagina één grote grid te maken
- Je kan dat doen werken, maar dit is nodeloos complex
- Beter is het om vanuit de paginabehoeftes te werken met meerdere, eenvoudiger te beheren grids

Webtech - CSS - les5 - kristof.michiels01@ap.be