CSS Grid

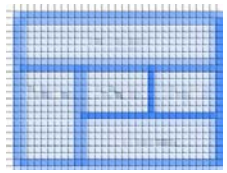
The page layout container for us all

Alumni presentation 4/29/2018

CENTRIQ TRAINING

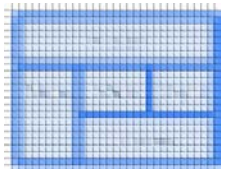
About

- Doug Hoff, Centriq Training
  - Java, JavaScript, C#, php
  - architecture, programming, analysis, testing, web frameworks
- dhoff@centriq.com
- Twitter: @doug_hoff
- Blog: http://socialitoutbursts.com/



Layout containers

Table



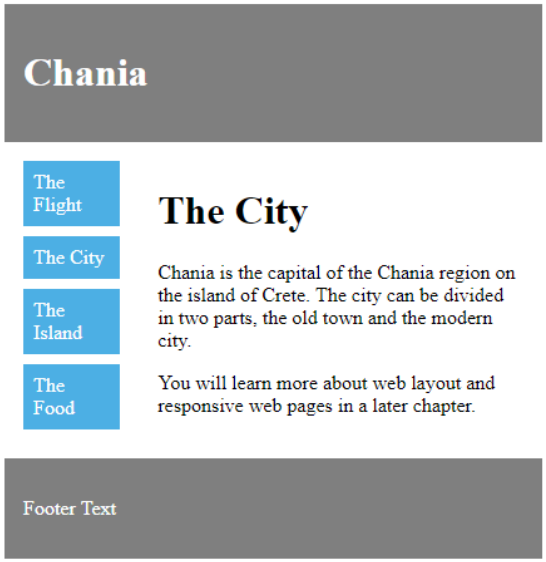
- Proper use
  - data formatting
  - static layout
- Technical debt
  - manual rearrangement
  - lots of CSS

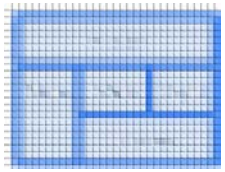
Firstname	Lastname	Age
Jill	Smith	50
Eve	Jackson	94
John	Doe	80

```
<style>
table, th, td {
    border: 1px solid black;
    border-collapse: collapse;
}
th, td {
    padding: 15px;
    text-align: left;
}
</style>
</head>
<body>
<table style="width:100%">
  <tr>
    <th>Firstname</th>
    <th>Lastname</th>
    <th>Age</th>
  </tr>
  <tr>
    <td>Jill</td>
    <td>Smith</td>
    <td>50</td>
  </tr>
  <tr>
    <td>Eve</td>
    <td>Jackson</td>
    <td>94</td>
  </tr>
```

Floating divs

- Proper use
  - smaller blocks
- Technical debt
  - learn float and clear
  - tied to flow of document





```
* {
    box-sizing: border-box;
}
.header, .footer {
    background-color: grey;
    color: white;
    padding: 15px;
}
.column {
    float: left;
    padding: 15px;
}
.clearfix::after {
    content: "";
    clear: both;
    display: table;
}
.menu {
    width: 25%;
}
.content {
    width: 75%;
}
.menu ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
```

Multiple layouts & media queries

- Proper use
  - Unresolvable compatible design issues
- Technical debt
  - complexity, maintainability

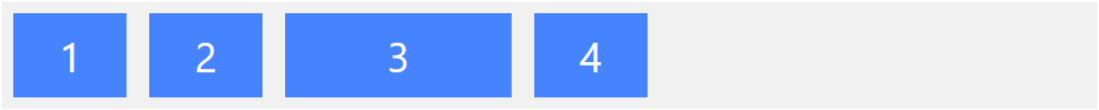

```
/* For desktop: */
.col-1 {width: 8.33%;}
.col-2 {width: 16.66%;}
.col-3 {width: 25%;}
.col-4 {width: 33.33%;}
.col-5 {width: 41.66%;}
.col-6 {width: 50%;}
.col-7 {width: 58.33%;}
.col-8 {width: 66.66%;}
.col-9 {width: 75%;}
.col-10 {width: 83.33%;}
.col-11 {width: 91.66%;}
.col-12 {width: 100%;}

@media only screen and (max-width: 768px) {
    /* For mobile phones: */
    [class*="col-"] {
        width: 100%;
    }
}
```

Flexbox

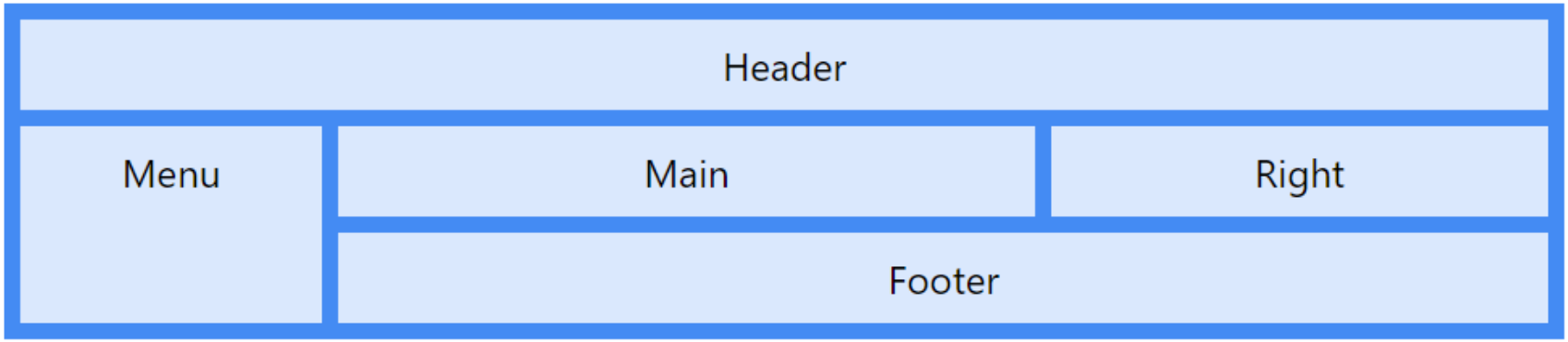
- give a container the ability to alter its items' width, height and order
- Proper use
  - one dimensional layout
  - menus
  - row of items

```
.flex-container {
  display: flex;
  height: 300px;
  justify-content: center;
  align-items: center;
}
```



CSS Grid

- In browsers March 2017
- best for 2-dimensional whole page layouts
- real grids not tables
- less media queries



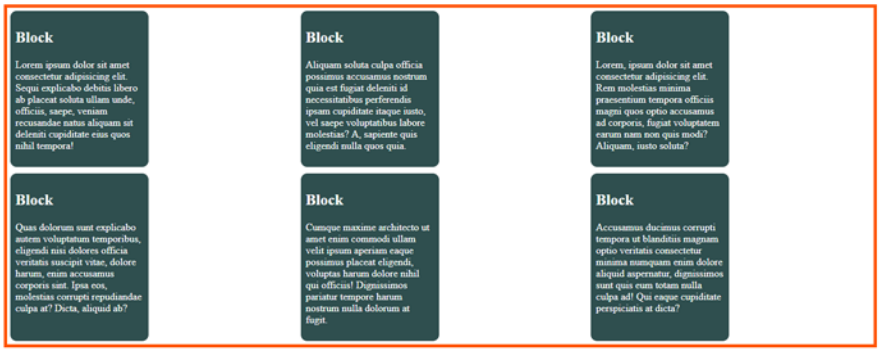
Compatibility

- All current browsers except
  - IE
  - UC Browser
- Applying other formatting will be discarded if grid can be applied
  - display: table-cell
  - flexbox
  - multi-column display

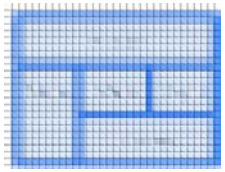
Grid HTML & CSS

Container

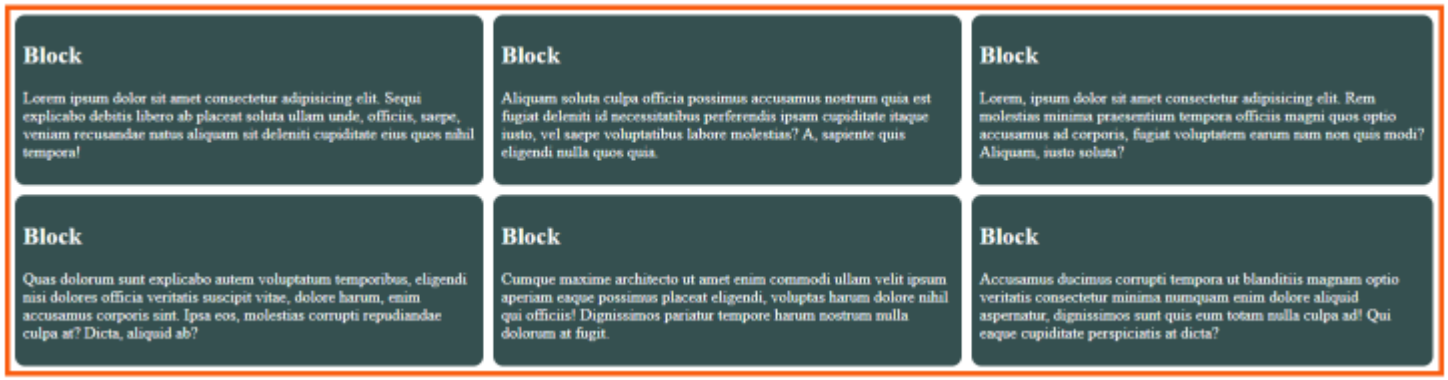
- Grid CSS (the container)
  - **display: grid;**
    - fixed-width items are spaced out
  - **display: inline-grid;**
    - container collapses around fixed-width items
  - subgrid removed to Level 2 – nested grid follows parent specs
  - display: contents – removes container box



Cells



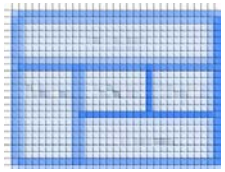
- Grid items (cells)
  - all children become grid items
  - items expand to fill column width if not constrained



Gaps

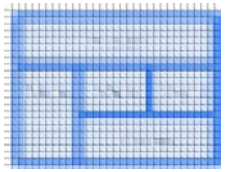
- Grid gap CSS
  - grid-column-gap #
  - grid-row-gap #
  - grid-gap cr#
  - grid-gap c# r#

Columns and rows

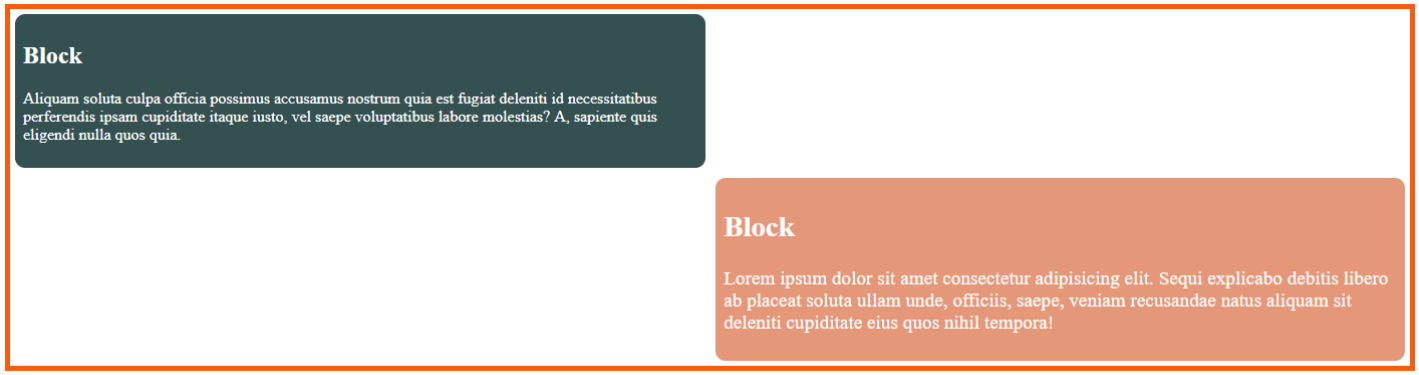


- Grid columns CSS – 4 columns
  - grid-template-columns: auto auto auto auto
  - grid-template-columns: # # # #
- Grid row height CSS – format as you need
  - grid-template-rows: 80px 200px;
  - other rows based on content

Item absolute placement - lines



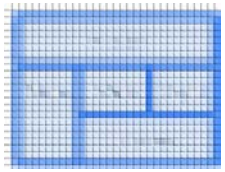
- three columns = 1 cell 2 cell 3 cell 4
- grid-row: 1; grid-column: 2;
- Spans
  - grid-row-start: 1; grid-row-end: 4;
  - grid-column-start: 2; grid-column-end: 4;



Item absolute placement - lines

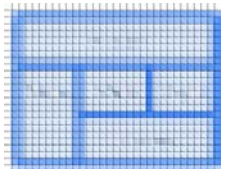
- Span shortcuts
  - grid-row: 1 / 4; grid-column: 2 / 4;
  - grid-row: 1 / span 3; grid-column: 2 / span 2;
- Shortcut
  - grid-area: 1 / 2 / 4 / 4

Item absolute placement - names



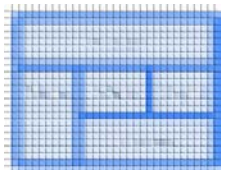
- Use names instead of numbers
- Name rows / columns
  - grid-template-columns: [c1] auto [c2] auto [c3] auto [c4 sidebar-column] auto [c-end];
  - grid-template-rows: [r1 header-row] auto [r2] auto [r3] auto [r4] auto [r-end];
- Multiple names are separated by spaces

Item absolute placement - names



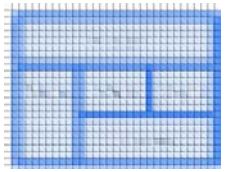
- Repeated names
  - repeat(7, [row-start] 1fr [row –end])
  - row-start **row** row-end row-start 2 **row** row-end 2 row...

fr



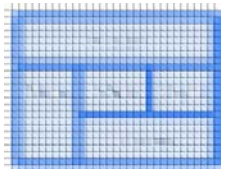
- fractional measurements divide up unused parts
  - 1fr 1fr 1fr 1fr
  - 1fr 2fr 3fr 1fr
  - 20% 2fr 3fr 1fr
  - 20% 40% 2fr 2fr
  - 200px 2fr 2fr 2fr

Columns and rows – repeat( )



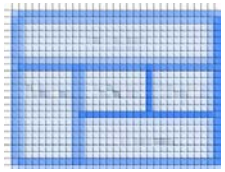
- repeat(# of times, units to use)
- grid-template-rows: repeat(…)
- grid-template-columns: repeat(…)
- grid-template-columns: repeat(3, 1fr)
- grid-template-columns: 2fr repeat(3, 1fr) 2fr

minimum / maximum sizing



- grid-template-rows, grid-template-columns
- minmax(minimum size of track, maximum size of track);
- auto = expand with content
- used as one size argument
- minmax(100px, auto) 2fr 1fr
- 1fr minmax(auto, 50%) 1fr

Responsive



- Media queries
- Start small
- Use min-width as you increase

```
/* Responsive layout */
/* smallest sizes first */
.responsive {
    grid-template-columns: auto;
    grid-template-areas: "header" "nav" "main" "aside" "footer";
}
@media screen and (min-width: 550px) {
    .responsive {
        grid-template-columns: minmax(20%, 200px) auto;
        grid-template-areas: "header header" "nav main" "aside aside" "footer
        footer";
    }
}
@media screen and (min-width: 767px) {
    .responsive {
        grid-template-columns: minmax(20%, 200px) auto minmax(20%, 200px);
        grid-template-areas: "header header header" "nav main aside" "nav main
        aside" "nav footer footer";
    }
}
```

	Queries targeting a maximum width.
	Queries targeting widths within a range.
	Queries targeting a minimum width.

Questions?

- Email me if you have any?