CSS Grid Layouts

Georg Niess, Arwin Roubal, Stefan Thurner, Enrique Barba Roque

Graz University of Technology
A-8010 Graz, Austria

30 Nov 2019

Abstract

This paper aims to introduce a state-of-the-art way to layout web pages in CSS, introducing CSS Grid. This work gives an introduction to CSS Grid and its most important principles, properties, features and options, showcasing why Grid is recommended to use compared to older techniques. Furthermore, this paper explores Grid Generators, including available tools online and evaluating their usefulness. This paper introduces the latest browser tools to examine the grid in the browser source inspectors. All things considered, this paper wants to give an overview of a fairly new topic in web page layout design with CSS.

Contents

List of Figures

List of Listings

Chapter 1

The History Of Layout In CSS

If one wants to understand which improvements CSS Grid brought and which issues it fixed, one first needs to understand which layout-techniques existed before it and what problems they had to cope with. Therefore, we want to give a quick introduction to historical methods to give an impression on the evolvement of layout options in CSS. It is worth to mention that those techniques are not all that 'historical' themselves as they are still widely used all across the web as CSS Grid is fairly new.

1.1 Float

The early layouts in CSS used Floats. The main content was often given a margin to have space for placing elements. In that time one could have elements on a website overflowing each other when increasing the font size for example. But what is Floating? It is a CSS Method of positioning elements by moving them as one would do with an image in Microsoft Word. If one uses the Float keyword in CSS the element gets moved to the desired direction [Andrew 2017].

The float property can have following values:

- left - movement to the left
- right - movement to the right
- none - default, element does not float
- inherit - inherit parent value

[W3School 2019]

Now it is not difficult to see that the options here are limited. In comparison with its successors, the weakness of float in regards to reordering elements becomes obvious. The strict dependence on HTML source code order with no instruments at hand to layout in CSS only developers had to use hacky tricks to achieve their results.

1.2 Flexbox

Many issues were fixed with the introduction of Flexbox. The so-called Flexible Box module brought features that were designed for responsive web. When using Flexbox one makes the items in the parent tag, called the flex container, stretch inside their container to adjust to screen size. These items are called flex items and can be reordered in CSS without interfering with the HTML source code. Items can be told to wrap into multiple lines with the "wrap" property.

However, one still needs a parent element inside the HTML that takes the role of the flex container. Another downside is that one lays out items either in column or row. One can not control both in a single Flexbox.

From this, it should be obvious that Flexbox is not designed for grid layouts. But in many cases, designers want to lay out in two dimensions. Once again, workarounds were the only possibility to reach desired results [Andrew 2017].

1.3 The New Way: CSS Grid

This is where CSS Grid comes in. The way of layouts in CSS Grid completely changes how Grid-based user interfaces can be designed. Up to the introduction of CSS Grid, all attempts at lay outing a two-dimensional page were basically hacks and failed to cover important functionality properly, like responsiveness for instance [CSS-TRICKS 2016]. It is easy to use and responsive by design too. And of course one can display page elements in any order, not depending in any way on the order described in source code. But it is worth noting that disconnecting the visual display from source order can be used for good and bad alike.

Chapter 2

Features of CSS Grid Module

The CSS Grid module has many useful features. These range from positioning to responsiveness and layout changes. This chapter will take a look at the most import ones and give examples for them.

2.1 The Ease Of Positioning

With CSS Grid designing works by defining a grid and populating the cells with content afterwards. One has many options on how to position content, there are many syntactical options, like properties that take line numbers and can also span the content across multiple cells. All of this is CSS only, and not dependent on any HTML code. Or one can use self-defined area names to assign content. There is an own property, the "grid-area" property for this purpose. It is easy to see how powerful Grid is. One can span across rows and columns, creating two-dimensional layouts [Andrew 2019].

The most basic one is to simply give a column start and endpoint definition and do the same for the row. A screenshot of the following examples is shown in Figure 2.1 and the corresponding HTML code can be seen in Listing 2.1.

```
grid-column-start: 2;
grid-column-end: 3;
grid-row-start: 1;
grid-row-end: 2;
```

Simple numbers are then used to choose the correct cell. This can be shortened further by using single "/" for separation of the start and endpoint.

```
#positioning-with-grid .b {
  grid-column: 2 / 3;
  grid-row: 2 / 3;
}
```

The user can also define aliases instead of using numbers if preferred.

```
#positioning-with-grid .c {
  grid-column: col 3;
  grid-row: row 2;
}
```

To stretch an element over more than one cell, the "span" keyword can be used.

```
#positioning-with-grid .d {
  grid-column: 1 / 2;
  grid-row: 1 / span 2;
}
```

```
<div class="wrapper">
  <div class="box a">A</div>
  <div class="box b">B</div>
  <div class="box c">C</div>
  <div class="box d">D</div>
  <div class="box e">E</div>
  <div class="box f">F</div>
</div>
```

Listing 2.1: This is the simple HTML code used in the examples of Section 2.1

Figure 2.1: This is a screenshot of the output of the examples of Section 2.1.

If space is a concern, the area can be defined in a single line sacrificing readability for space. The order of the numbers is "row-start", "column-start", "row-end", and "column-end".

```
#positioning-with-grid .f {
  grid-area: 1 / 3 / 2 / 4;
}
```

2.2 Automatic Placement

Not every element of a Grid must have a clear position defined. The way this is handled is by simply filling up free space in the Grid layout with those undefined elements. This allows for dynamic content placement and ordering without any intervention [Andrew 2019]. A screenshot of the output is shown in Figure 2.2 and the corresponding HTML code in Listing 2.2.

2.3 Repeat Template

Columns and rows can also be defined by using the repeat keyword. This creates a space- and time-saving method to place several rows or columns next to each other with the same size [Andrew 2019]. The corresponding CSS code can be seen in Listing 2.3. A screenshot of the output is shown in Figure 2.2 and the corresponding HTML code in Listing 2.2.

2.4 Absolute Positioning

Elements can also be positioned absolutely in a CSS Grid. This is done by choosing an anchor position that the element is grounded to. The attribute in question is "position: absolute"; the anchor can then be chosen with for example the "top", "bottom", "left" or "right" properties [Andrew 2019]. The

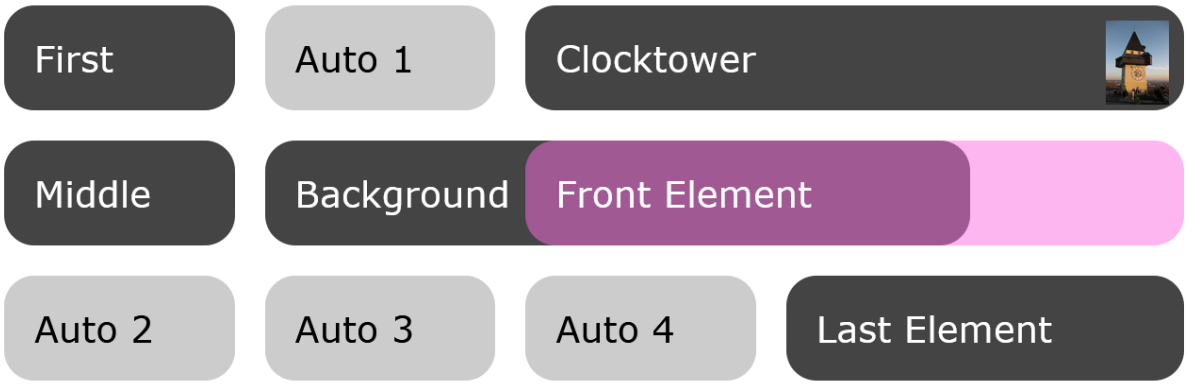

Figure 2.2: This screenshot shows various features of Grid like absolute positioning and layering.

```
<div class="wrapper">
  <div class="box a">First</div>
  <div class="box b">Clocktower <br /><img src="Clocktower.jpg" alt="clocktower.jpg"
       class="clocktower" /></div>
  <div class="box c">Middle</div>
  <div class="box d">Background</div>
  <div class="box e">Last Element</div>
  <div class="box f">Front Element</div>
  <div class="box g">Auto 1</div>
  <div class="box h">Auto 2</div>
  <div class="box i">Auto 3</div>
  <div class="box j">Auto 4</div>
</div>
```

Listing 2.2: This is the simple HTML code used in various examples.

```
#grid-repeat .wrapper {
  ...
  grid-template-columns: repeat(5, [col] 12rem ) ;
  grid-template-rows: repeat(3, [row] auto  );
}
```

Listing 2.3: In this example, a template is created with the repeat keyword allowing for a quick formal layout.

```
#grid-layering .clocktower {
  ...
  position: absolute;
  top: 1rem;
  right: 1rem;
}
```

Listing 2.4: An Example of an image being positioned absolutely in a CSS Grid.

```
<div class="wrapper">
  <div class="box header">Header</div>
  <div class="box sidebar">Sidebar</div>
  <div class="box sidebar2">Sidebar 2</div>
  <div class="box content">Content
    Here is some long text that will fill this pretty broad element with stuff.</div
        >
  <div class="box footer">Footer</div>
</div>
```

Listing 2.5: This is the simple HTML code used for the responsive examples.

corresponding CSS code can be seen in Listing 2.4. A screenshot of the output is shown in Figure 2.2 and the corresponding HTML code in Listing 2.2.

2.5 Grid Area

Positioning can also be done using areas. Instead of just using numbers for enumerating, fixed names can be created with the "grid-template-areas" attribute. The names are assigned simply by laying out the aliases in a Grid-like way which is very intuitive. Areas can also span over multiple cells [Andrew 2019]. The corresponding CSS code can be seen in Listing 2.6. A screenshot of the output is shown in Figure 2.5 and the corresponding HTML code in Listing 2.5.

2.6 Template Columns and Rows

In CSS Grid, instead of using pixels or rems, there are additional ways to define the size of columns or rows. Size can be defined using percentages of the available space, or by using the auto attribute which fills up the remaining space. Fractions can also be used instead to divide the space into several parts [Andrew 2019]. The corresponding code for alle these features can be seen in Listing 2.7. A screenshot of the output is shown in Figure 2.3 and the corresponding HTML code in Listing 2.5.

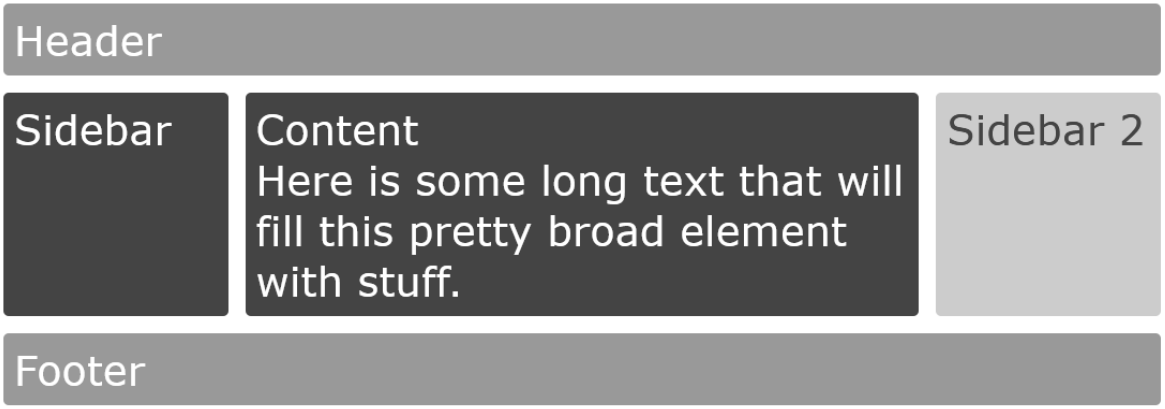

Figure 2.3: This screenshot shows a Grid example in desktop layout.

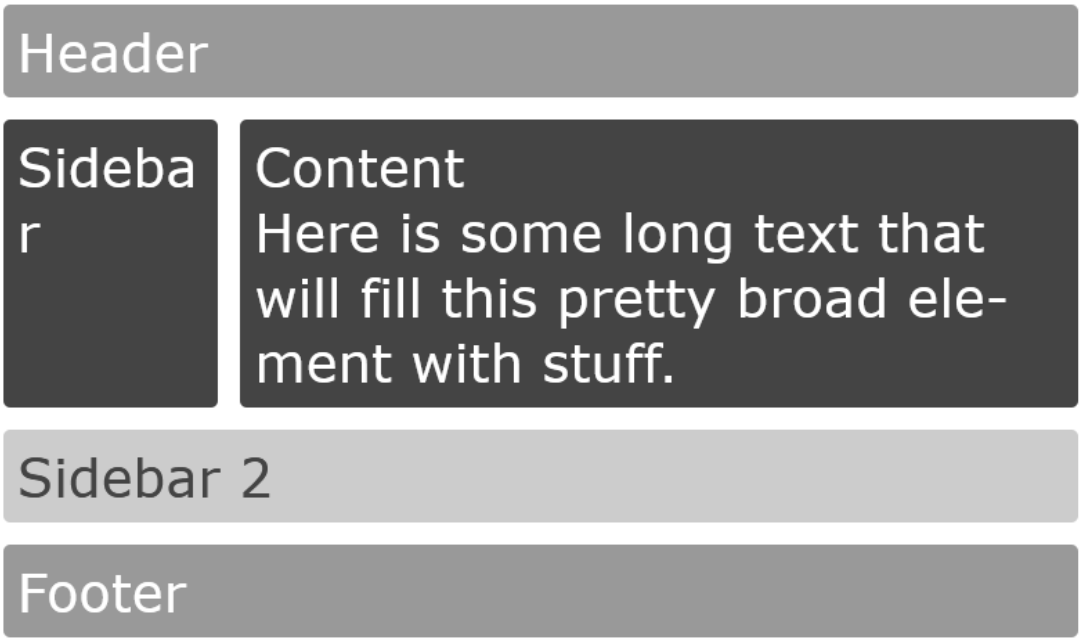

Figure 2.4: This screenshot shows a Grid example in tablet layout.

```
#example .wrapper {
  grid-template-areas:
  "header   header"
  "sidebar  content"
  "sidebar2 sidebar2"
  "footer   footer";
}
```

Listing 2.6: Creating Areas is very intuitive and straightforward. The general Grid is simply laid out by organizing the titles of the cells. To add an element, only the alias needs to be added.

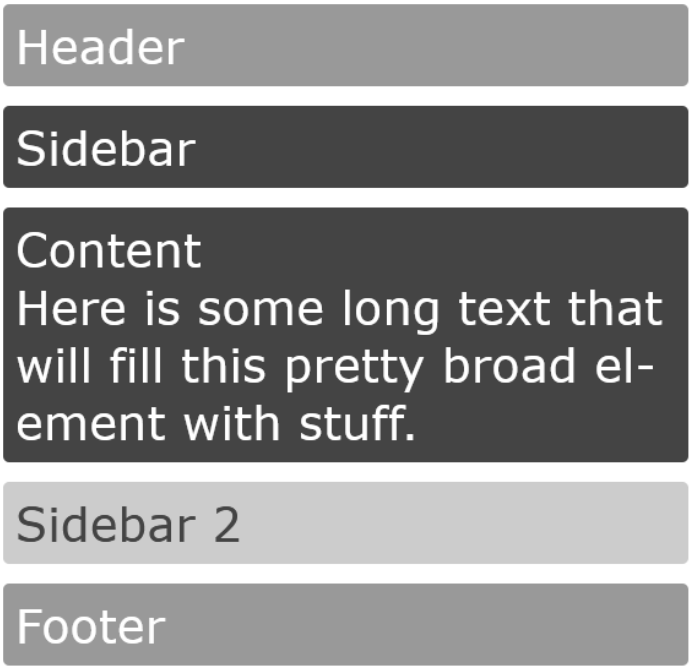

Figure 2.5: This screenshot shows a Grid example in mobile layout.

```
#example-template .wrapper {
  grid-template-columns: 1fr 3fr 1fr;
  grid-template-rows: 20% 40% auto;
  grid-template-areas:
  "header  header  header"
  "sidebar content sidebar2"
  "footer  footer  footer";
}

.header {
  grid-area: header;
}
```

Listing 2.7: Creating Areas is very intuitive and straightforward. The general Grid is simply laid out by writing the aliases of the cells. To then add an element to a cell, only the alias needs to included in its "grid-area" tag.

```
#media-queries .wrapper {
  display: grid;
  grid-template-areas:
    "header"
    "sidebar"
    "content"
    "sidebar2"
    "footer"
}

@media only screen and (min-width: 40rem)  {
  #media-queries .wrapper {
    grid-template-columns: 1fr 3fr;
    grid-template-areas:
    "header   header"
    "sidebar  content"
    "sidebar2 sidebar2"
    "footer   footer";
  }
}

@media only screen and (min-width: 60rem)   {
  #media-queries .wrapper {
    grid-template-columns: 1fr 3fr 1fr;
    grid-template-areas:
    "header  header  header"
    "sidebar content sidebar2"
    "footer  footer  footer";
  }
}
```

Listing 2.8: This is an example of a layout having three different versions for different screen sizes using media queries.

2.7 Media Queries and CSS Grid

Media Queries can be used to check the screen size of the client and to make the layout responsive. This is done by checking for min-width and then defining a new Grid layout. The default layout is simply overwritten by the new one. This allows for drastic layout and even content order changes in a very easy way. Websites can be fitted to different devices like mobile and desktop without much work or any changes in the HTML. The corresponding code can be seen in Listing 2.8. A screenshot of the output is shown in Figure 2.3, 2.4 and 2.5, and the corresponding HTML code in Listing 2.5.

2.8 Layering

Elements of a Grid can be layered on top of each other with the property "z-index". This adds a third-dimension to layouts. The furthest back position would be at zero, and each increase by one is one layer further to the front. This can also be nicely combined with transparencies. A screenshot of the output is shown in Figure 2.2 and the corresponding HTML code in Listing 2.2.

```
#grid-layering .f {
  z-index: 20;
}
```

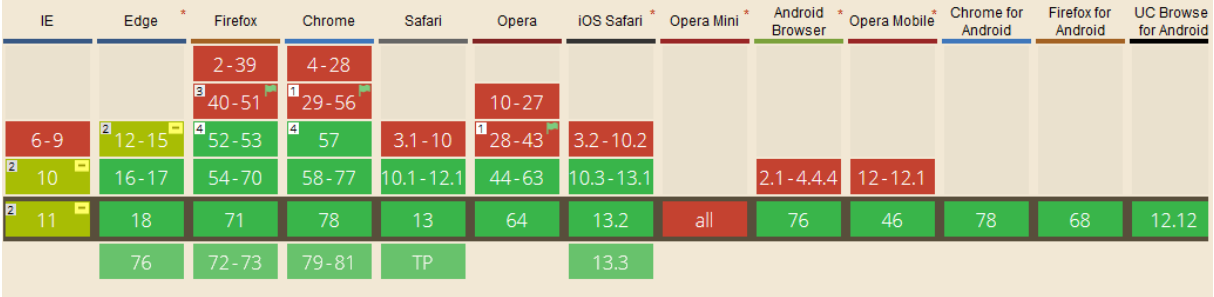

Figure 2.6: This image shows the combatibility of different browsers with Grid [Can I use 2019].

This element is positioned "at z-index" 20, so it could have up to 20 different layers below it.

2.9 Browser Support

There is support for Grid in Chrome, Firefox and Safari since March 2017. Edge was a bit late but shipped support October 2017. There are of course still older browsers that do not support grid [Andrew 2019]. But it really should not be an issue. First of all, the userbase of today is much more likely to use up to date browsers and get regular updates than data from ten years ago would suggest. One has to make sure to make the right assumptions when talking numbers here. And new CSS might become usable far faster than one would expect when starting a project [Andrew 2017].

Furthermore, old browsers will simply ignore properties in CSS that they do not understand. This means that one only needs to add feature queries to properties that are not exclusive to CSS Grid to ensure support on older browsers. In most cases, one will only need to overwrite properties such as width or margin. Support for feature queries themselves is also very good [Matuzović 2017].

An overview of Grind compatibility is shown in Figure 2.6

Chapter 3

CSS Grid Generators

In former chapters, CSS Grid was discussed in general. How it developed over time from CSS Float to more modern approaches. The workflow has been shown, different examples about how to use it efficiently were presented and CSS Grid has been compared to older methods like Float and Flexbox. Another part of this report was the discussion of Grid debugging for different Browsers, which should help web developers working with the Grid. The last topic of this paper will be a discussion and comparison of different Grid generating tools.

3.1 Introduction

Grid generators are a graphical tool for web developers. They are a convenient tool for creating a web page layout with CSS Grid and offer an intuitive and interactive method for designing a basic Grid layout. Instead of writing CSS and HTML code in separate files, such generators support the creation of a basic layout graphically and translate it to HTML and CSS code. This way design decisions can be tested and evaluated before applying to a real web page. Manipulating elements in these graphical tools is also very easy and intuitive. When the result is acceptable, the corresponding CSS and HTML code can be downloaded for further operational steps. After this short theoretical introduction, the next logical step is to present some selected tools which illustrate the now discussed properties.

3.2 CSS Grid Generator by Sarah Drasner

This first discussed generator offers basic but efficient functionalities for creating a first CSS Grid layout [Drasner 2019]. A screenshot of the interface is shown in Figure 3.1 and the corresponding code can be seen in Listing 3.1. Numbers of columns and rows can be defined and sized with fractions. Column and row gaps can also be included using pixels. After a basic grid is defined, elements can be added to it by clicking the corresponding areas in the graphical representation. It is also possible to create elements which stretch across multiple rows or columns by dragging across them. However, this method can get ponderous fast and overlapping elements may confuse inexperienced Grid users. Another weakness is that the grid areas and the elements within it cannot be named directly inside the tool, it is necessary to change this manually in the corresponding code. Aside from this basic functionality, CSS Grid Generator does not offer any other more complex functions, but it is a great starting point for getting introduced to CSS Grid layout and for learning basic use cases. Another advantage is that it is available in offline mode, so it is possible to save this website locally.

This tool is still new and aside from some basic functions, it does not offer much now. Nevertheless, being an open-source software in active development, more features may get added in the future and complex functionalities might find their way into this project.

```
HTML:

<div class="parent">
<div class="div1"> </div>
<div class="div2"> </div>
<div class="div3"> </div>
<div class="div4"> </div>
<div class="div5"> </div>
</div>


CSS:

.parent {
display: grid;
grid-template-columns: 1fr 4fr 1fr;
grid-template-rows: 1fr 4fr 1fr;
grid-column-gap: 0px;
grid-row-gap: 0px;
}
.div1 { grid-area: 2 / 2 / 3 / 3; }
.div2 { grid-area: 1 / 1 / 2 / 4; }
.div3 { grid-area: 3 / 1 / 4 / 4; }
.div4 { grid-area: 2 / 1 / 3 / 2; }
.div5 { grid-area: 2 / 3 / 3 / 4; }
```

Listing 3.1: Example code generated by CSS Grid Generator for corresponding layout shown in Figure 3.1.

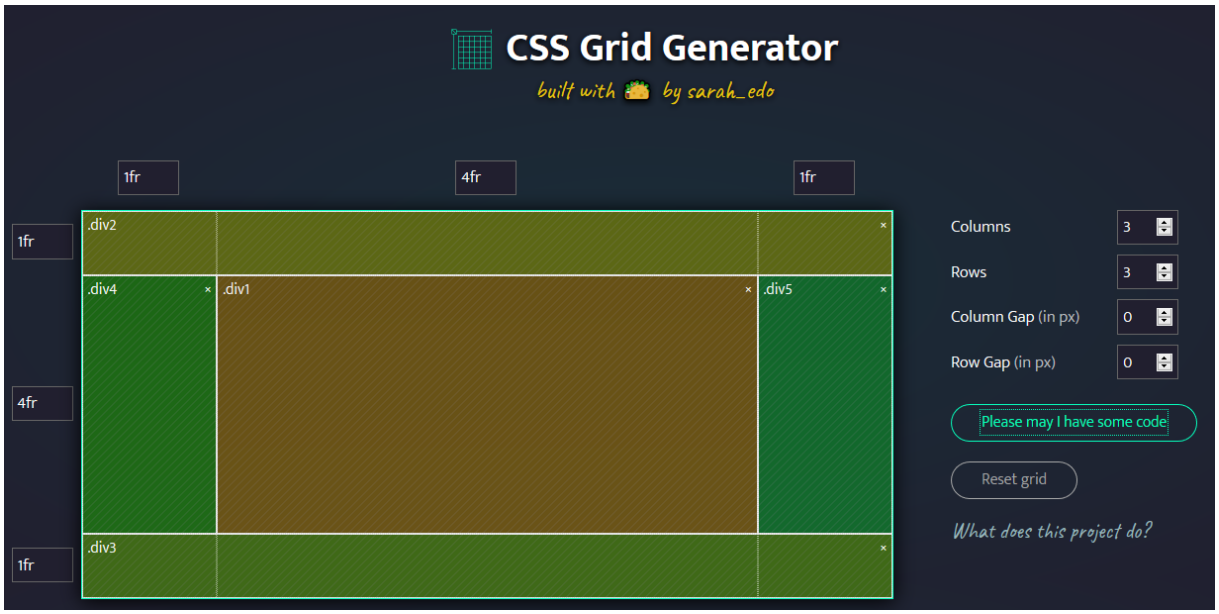

Figure 3.1: Interface of CSS Grid Generator by Sarah Drasner

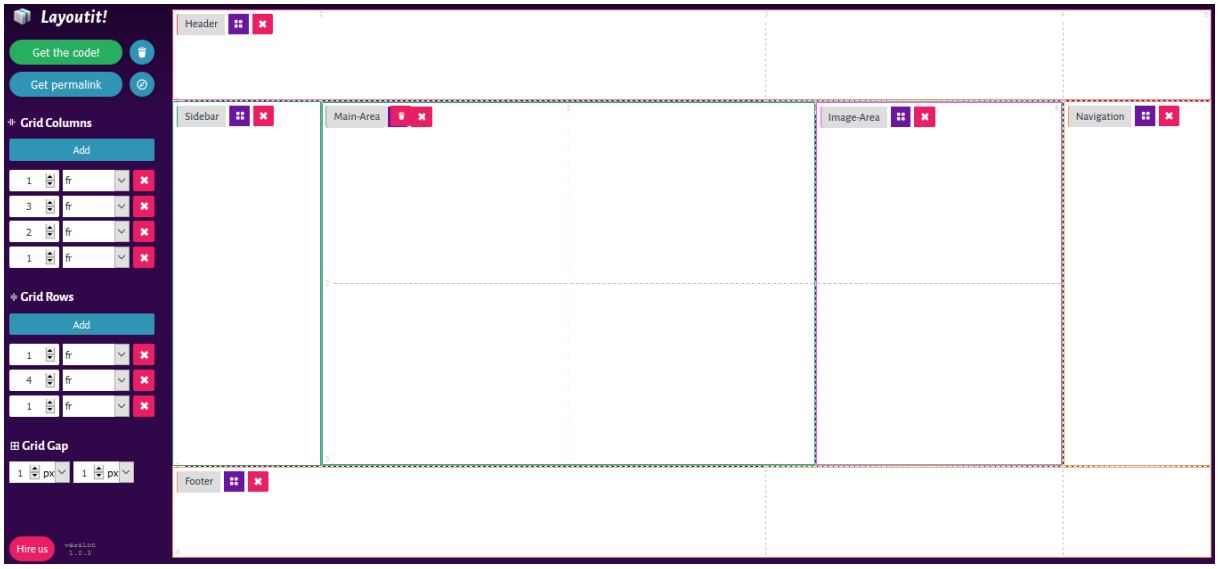

Figure 3.2: Interface of Layoutit! by Leniolabs

3.3 Layoutit! By Leniolabs

Layoutit! is another very intuitive tool for creating a Grid layout [Leniolabs LLC 2019]. It offers some more complex functions and works a more intuitive way as the CSS Grid Generator. A screenshot of the interface can be seen in Figure 3.2 and the corresponding code is shown in Listing 3.2. The basic functionality is very similar to the CSS Grid Generator by Sarah Drasner. Rows and columns can be defined in the sidebar, which can be sized with fractions, "Like in the first tool, elements can be added by simply clicking and dragging inside the graphical area. However, these tools work much more natural, solid and less vulnerable for bugs.

This tool also offers some advantages compared to the last one. For instance, it uses Grid areas to describe the general layout, which enables more control over the page design, as already described in former chapters of this paper. Layoutit! is also able to create more complex designs using nested Grids. As the CSS Grid Generator by Sarah Drasner, Layoutit! is also available in offline mode. Overall Layoutit! offers some handy features which really enhances the process of designing a general page layout. It works in a more intuitive way and it feels solid in its functionality.

3.4 Vue Grid Generator by Masaya Kazama

The third and last tool which should be discussed is the Vue Grid Generator created by Masaya Kazama [Kazama 2019]. It is yet another tool for generating Grids in a graphical way. A screenshot of the interface is presented in Figure 3.3 where the corresponding code is shown in Listing 3.3.

Created with the library vue.js, this generator is very similar to Layoutit! concerning its functionality. The Grid layout is defined by adding elements vertically or horizontally and their size can be further described with fractions, "A unique feature of this tool is the possibility to use presets and to import or export layouts directly. It is also very useful that the HTML code, as well as the CSS code, are displayed next to the graphical representation. However, it is not synchronized in both directions, which means that the presented code cannot be manipulated directly.

Overall the Vue Grid Generator is a good tool with the most features out of the three here discussed generators. Plus, it offers some useful functions, that could be interesting depending on personal preferences and desires. A little downside is the fact, that it does not work in offline mode. However, the newest version of the build can be downloaded on the GitHub Account of Masaya Kazama.

```
HTML:

<div class="grid-container">
  <div class="Header"></div>
  <div class="Main-Area">
    <div class="Text-Area"></div>
    <div class="Image-Area"></div>
  </div>
  <div class="Sidebar"></div>
  <div class="Navigation"></div>
  <div class="Footer"></div>
</div>


CSS:

.grid-container {
  display: grid;
  grid-template-columns: 1fr 3fr 2fr 1fr;
  grid-template-rows: 1fr 4fr 1fr;
  grid-template-areas: "Header Header Header Header" "Sidebar Main-Area
      Main-Area Navigation" "Footer Footer Footer Footer";
}

.Header { grid-area: Header; }

.Main-Area {
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 1fr 1fr;
  grid-template-areas: "Text-Area Text-Area Image-Area" "Text-Area Text-
      Area Image-Area";
  grid-area: Main-Area;
}

.Text-Area { grid-area: Text-Area; }

.Image-Area { grid-area: Image-Area; }

.Sidebar { grid-area: Sidebar; }

.Navigation { grid-area: Navigation; }

.Footer { grid-area: Footer; }
```

Listing 3.2: Example code generated by Layoutit! for corresponding layout shown in Figure 3.2.

```
HTML:

<div class="container">
  <div class="header">header</div>
  <div class="navbar">navbar</div>
  <div class="left">left</div>
  <div class="main">main</div>
  <div class="right">right</div>
  <div class="footer">footer</div>
</div>


CSS:

.container {
  display: grid;
  width: 100%;
  height: 100%;
  grid-template-areas: "header header header"
  "navbar navbar navbar"
  "left main right"
  "footer footer footer";
  grid-template-columns: 1fr 4fr 1fr;
  grid-template-rows: 1fr 1fr 4fr 1fr;
}
.container > div {
  border: 1px dashed #888;
}

.header {
  grid-area: header;
}
.navbar {
  grid-area: navbar;
}
.left {
  grid-area: left;
}
.main {
  grid-area: main;
}
.right {
  grid-area: right;
}
.footer {
  grid-area: footer;
}
```

Listing 3.3: Example code generated by Vue Grid Generator for corresponding layout shown in Figure 3.3.

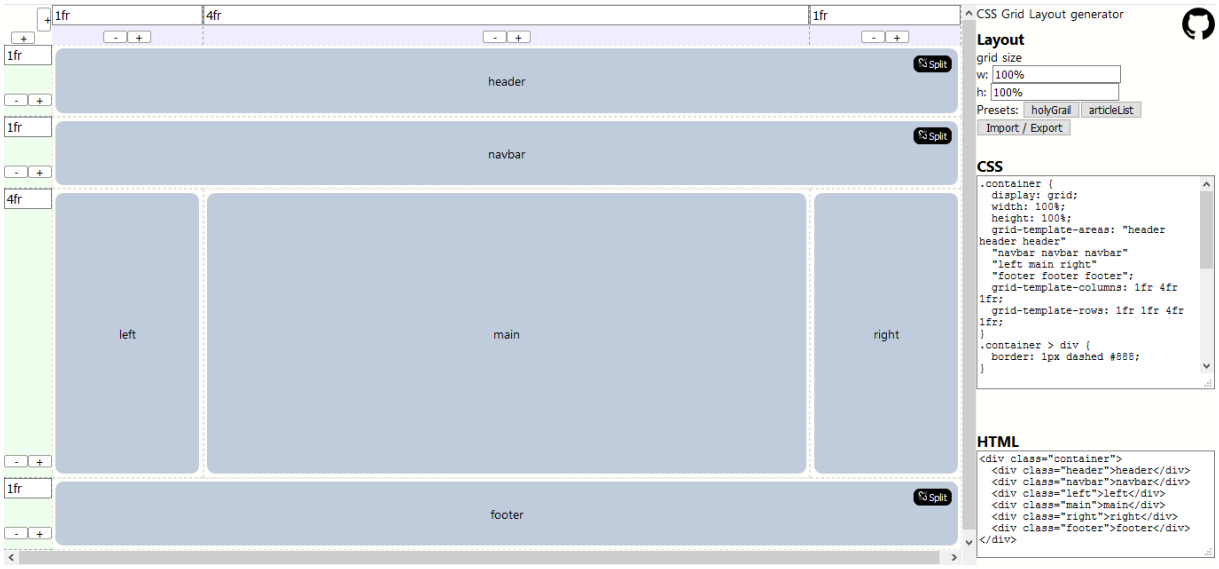

Figure 3.3: Interface of Vue Grid Generator by Masaya Kazama

3.5 Conclusion

Starting with CSS Grid can be irritating when older approaches of CSS are still omnipresent. The advantages of these new methods were discussed in this paper and Grid generators are a great introduction to CSS Grid in general. Most of the generator tools are very similar in their usability and functionality, they differ just in minor features. Some are more intuitive concerning their graphical representation, some offer additional features as presets. Which one of the tools should be recommended depends on the individual needs of a web developer. Overall CSS Grid generators are a good way to experiment with CSS Grid they are a great tool for inexperienced users. However, their restrictive possibilities make these tools irrelevant for veteran web developers and more experienced users.

Chapter 4

Grid Debugging Tools

This chapter will take a look at inspector tools some browsers offer for working and debugging the Grids built for a website, and their different features will be studied and compared.

To the date, only Mozilla Firefox and Google Chrome have some sort of feature in their inspectors related to Grids, albeit there is also a Chrome extension for Grid inspection that will be studied too.

4.1 Firefox Grid Inspector

Firefox has the inspector with the most features for debugging Grids [Mozilla 2019]. It can be accessed from the standard inspector interface, where a tab for Grid inspection can be found under the section "Layout". There, a list for the containers with the property "display:grid" on the page can be found. The containers selected in that list will be highlighted with the desired colours as seen in Figure 4.1. Then, multiple display options can be selected:

- Display line numbers, which shows the columns and row numbers that would apply for establishing the span of the elements in the properties "grid-column" and "grid-row".
- Display area names, that would display the name of the areas over them if these were defined.
- Extend lines infinitely, property that extends the row and column lines to the end of the viewport in order to visualize possible extensions for the Grid or alignment with other elements.

It is also important to notice that these display options will apply for all the currently selected Grids, with no possibility to choose different configurations for each Grid.

Additionally, when only one Grid is selected for display, the inspector will also show a diagram of the Grid, and hovering over a Grid area in it, the area from the actual Grid will be highlighted.

More importantly, any changes done to the CSS properties of the elements through the inspector will also reflect in the Grid section of it. This allows the immediate visualization of modifications in the Grid.

The Grid panel was added in version 56 of Firefox, released in September 2017.

4.2 Chrome Grid Inspector

This inspector offers more basic features. For this browser, when the code of an element of the Grid is selected through the inspector, the Grid that contains it will be marked with dotted lines as shown in Figure 4.2.

This functionality was added to the inspector with Chrome v62 in October 2017.

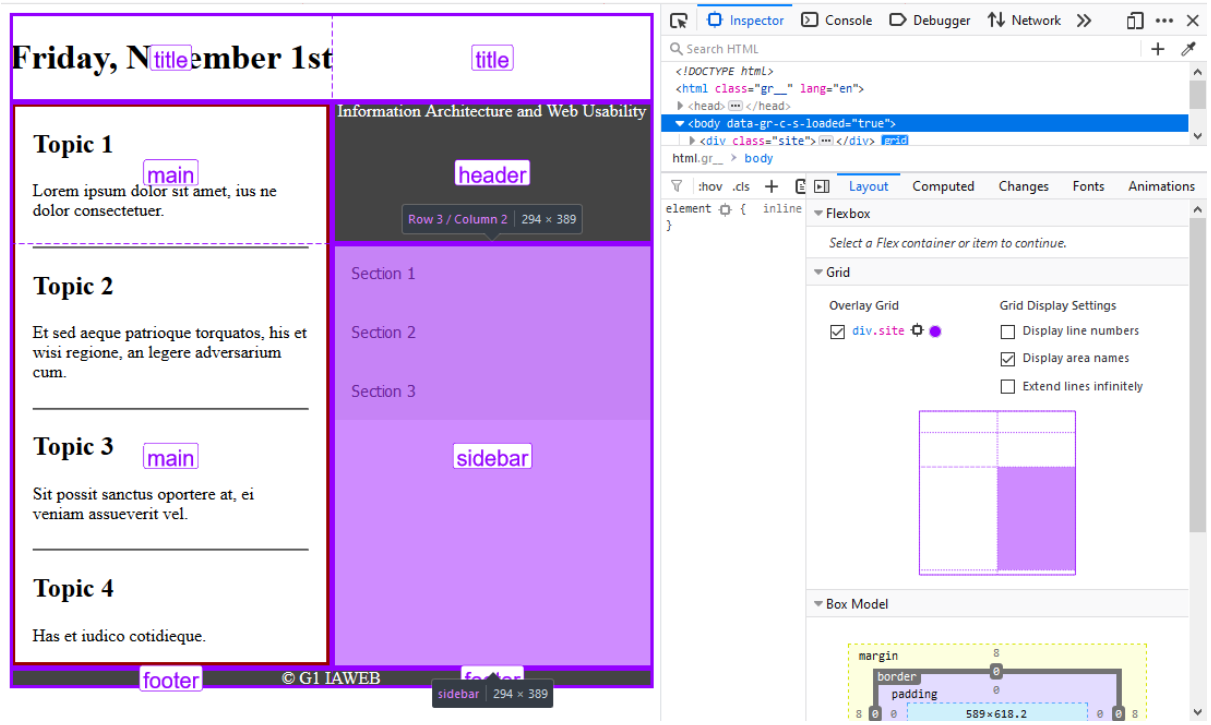

Figure 4.1: Firefox Grid Inspector

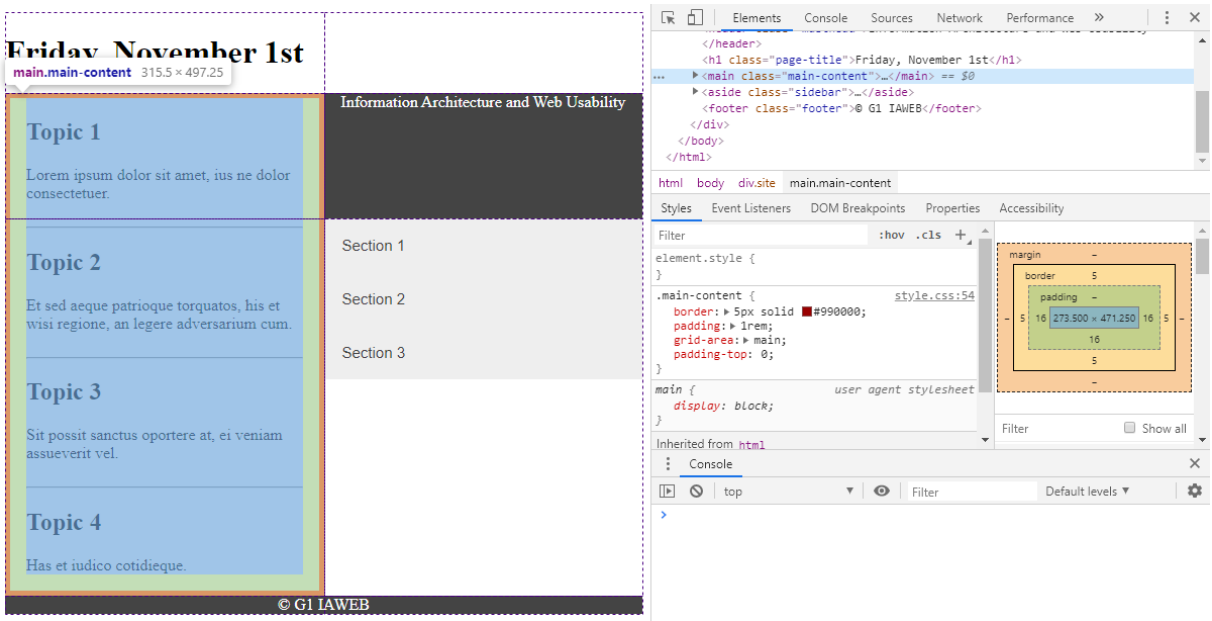

Figure 4.2: Chrome Grid Inspector

Figure 4.3: Gridman Grid Inspector

4.3 Gridman Grid Inspector

Gridman is an extension for Google Chrome that adds a grid inspector functionality [Anton Savinskiy 2019]. When active, the Grid selected by the mouse will be highlighted with red lines, similar to the Chrome Grid inspector. A screenshot of this extension can be seen in Figure 4.3.

This extension was released before Chrome v62 (the Chrome Web Store does not show the release date, but the oldest review is from August 2017), so, when the Chrome inspector got released, this extension got deprecated.

Bibliography

Andrew, Rachel [2017]. *The New CSS Layout*. A Book Apart, 02 Oct 2017. ISBN 1-93-755768-5 (cited on pages 1–2, 10).

Andrew, Rachel [2019]. *Grid by Example - Everything you need to learn CSS Grid Layout*. 01 Dec 2019. https://gridbyexample.com/ (cited on pages 3–4, 6, 10).

Anton Savinskiy [2019]. *Gridman - CSS Grid inspector. Ultra Fast!* 01 Dec 2019. https://chrome.google.com/webstore/detail/gridman-css-grid-inspecto/cmplbmppmfboedgkkelpkfgaakabpicn (cited on page 19).

Can I use [2019]. *CSS Grid Layout (level 1)*. 01 Dec 2019. https://caniuse.com/#feat=css-grid (cited on page 10).

CSS-TRICKS [2016]. *A Complete Guide to Grid*. 06 Nov 2016. https://css-tricks.com/snippets/css/complete-guide-grid/ (cited on page 2).

Drasner, Sarah [2019]. *CSS Grid Generator*. 01 Dec 2019. https://cssgrid-generator.netlify.com/ (cited on page 11).

Kazama, Masaya [2019]. *CSS Grid Layout*. 01 Dec 2019. https://github.com/miyaoka/grid-generator (cited on page 13).

Leniolabs LLC [2019]. *Layoutit*. 01 Dec 2019. https://grid.layoutit.com/ (cited on page 13).

Matuzović, Manuel [2017]. *Progressively Enhancing CSS Layout: From Floats To Flexbox To Grid*. Smashing Magazine. 24 Jul 2017. https://smashingmagazine.com/2017/07/enhancing-css-layout-floats-flexbox-grid/ (cited on page 10).

Mozilla [2019]. *Introduction to CSS Grid Layout - Firefox DevTools*. 01 Dec 2019. https://mozilladevelopers.github.io/playground/css-grid/03-firefox-devtools/ (cited on page 17).

W3School [2019]. *CSS Layout - float and clear*. 01 Dec 2019. https://css-tricks.com/snippets/css/complete-guide-grid/ (cited on page 1).