

NEW CSS3 LAYOUT MODES, HISTORY INTERFACE AND WEB WORKERS

Aki Huttunen

Thesis
May 2014

Degree Program in Software Engineering
Technology, communication and transport



JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES



Author(s) Huttunen, Aki	Type of publication Bachelor's Thesis	Date 14.05.2014
	Pages 49	Language English
		Permission for web publication (X)
Title New CSS3 layout modes, History interface and Web Workers		
Degree Programme Software Engineering		
Tutor(s) Peltomäki, Juha		
Assigned by Qvantel Business Solutions Oy		
Abstract <p>The objective of this thesis was to research and study existing and upcoming CSS3 and HTML5 technologies. The technology selected was Flexbox layout which provides an elegant system with HTML elements inside a container in rows that will 'flex' to fill the available space in different ways specified using CSS properties. Another technology was the grid layout that splits the page into rows and columns and allows content items to fill the grid built from those rows and columns. The third technology was History interface which enables the creation of new and the edition of current browser history events. Lastly, Web Workers were studied that allow the offloading of computation work into the background into another thread so that they will not block the main DOM thread.</p> <p>The study describes the basic concepts of these new technologies and their use through examples and cases.</p> <p>This thesis describes a field that does not contain many previous research projects; therefore the examples provided by this study are of value to web developers.</p>		
Keywords Flexbox layout, Grid layout, History interface, Web Worker, Responsive layout		
Miscellaneous		



Tekijä(t) Huttunen, Aki	Julkaisun laji Opinnäytetyö	Päivämäärä 14.05.2014
	Sivumäärä 49	Julkaisun kieli Englanti
		Verkojulkaisulupa myönnetty (X)
Työn nimi Uudet CSS3 layout moodit, historia rajapinta ja web workkerit.		
Koulutusohjelma Ohjelmistotekniikan koulutusohjelma		
Työn ohjaaja(t) Peltomäki, Juha		
Toimeksiantaja(t) Qvantel Business Solutions Oy		
Tiivistelmä <p>Opinnäytetyössä tarkasteltiin olemassa olevia ja tulevia CSS3 ja HTML5 tekniikoita. Näistä valittiin Flexbox layout joka tarjoaa elegantin tavan laittaa HTML elementtejä säiliön sisälle riveissä, joidenka kokoa voidaan venyttää erilaisin tavoin joita kontrolloidaan CSS ominaisuuksilla. Grid layout joka jakaa sivun sarakkeisiin ja riveihin joista rakennetaan ruudukko, joka voidaan täyttää HTML elementeillä. History interface joka mahdollistaa uusien historia tapahtumine muokkaamisen ja tämänhetkisen tapahtuman muokkaamisen. Ja viimeisenä Web Workkerit jotka mahdollistavat laskutyön siirtämisen taustalla pyörivään säikeeseen, jolloinka laskutyö ei pysty lukitsemaan DOM säiettä.</p> <p>Työssä käydään läpi eri tekniikoiden perusteet ja käydään esimerkkiä hyväksi käyttäen läpi niiden toiminta.</p> <p>Työn tuloksena saatiin tutkimus sellaiselle alueella jolle ei aikaisemmin ole tehty kovinkaan montaa tutkimusta. Työn tuloksena saatiin aikaan ohjeet ja esimerkit joidenka avulla kehittäjä voivat lähestyä näitä tekniikoita helpommin.</p>		
Avainsanat (asiasanat) Flexbox layout, Grid layout, History interface, Web Worker, Responsive layout		
Muut tiedot		

Contents

1	Introduction	7
2	Web Workers	8
2.1	Concepts	8
2.2	Dedicated Worker example	8
2.3	Conclusions	11
3	History interface	11
3.1	General	11
3.2	Adding new history events	12
3.3	Changing current history state	13
3.4	Multistep form example using pushState	13
3.5	Example of updating current session state	16
3.6	Conclusions	19
4	Flexbox layout	19
4.1	Flexbox concepts	20
4.2	Basic layout and wrapping	21
4.3	Ordering	27
4.4	Alignment	28
4.4.1	Alignment with auto margins	28
4.4.2	Main axis alignment	29
4.4.3	Cross axis alignment	30

	2
4.4.4 Alignment of item lines	31
4.4.5 Example	32
4.5 Conclusions	37
5 Grid layout	38
5.1 General	38
5.2 Concepts	39
5.3 Example	40
5.4 Conclusions	47
6 Summary	47
References	49

List of Figures

1	Calculating primes without a Web Worker. Does not even render HTML due to calculation being done inside head element	10
2	Calculating primes with a Web Worker	10
3	Order creation default form created by Ruby on Rails default form styles	14
4	Product selection step	15
5	Contact information step	15
6	Payment information and order saving	16
7	User comes to web page	17
8	User scrolls down to Article 2	17

9	Illustration of the Flexbox concepts when the direction is 'row' (originally from CSS Flexible Box Layout Module Level 1 W3C Working Draft)	20
10	Without Flexbox	22
11	With simple Flexbox layout	23
12	Illustration of how flex-basis. Red and blue numbers are flex-grow factors. (originally from CSS Flexible Box Layout Module Level 1 W3C Working Draft)	23
13	Flexbox automatically filling the container with its content	24
14	With green element having bigger grow factor	24
15	With 500px wide window	25
16	With 380px wide window	26
17	With 500px wide window. Wrapping with green element having bigger grow factor	26
18	With the blue third (blue) element ordered to be first	28
19	Navigation bar using Flexbox layout	29
20	Navigation bar with auto margin	29
21	Illustration of the different main axis alignment modes. (originally from CSS Flexible Box Layout Module Level 1 W3C Working Draft)	30
22	Illustration of the different cross axis alignment options. (originally from CSS Flexible Box Layout Module Level 1 W3C Working Draft)	31
23	Illustration of the alignment between lines. (originally from CSS Flexible Box Layout Module Level 1 W3C Working Draft)	32
24	Illustration of the transition of the layout from large screen to small screen	33
25	Layout before Flexbox layout	34

26	Flexbox layout on a wide screen	35
27	The layout of the example on a narrow screen	37
28	Horizontal game layout	38
29	Vertical game layout	39
30	Basic layout with simple styles	41
31	Horizontal layout with red lines showing row splits and blue lines showing column split	41
32	Vertical layout with red lines showing row splits and blue lines showing column split	42
33	Game horizontal layout with grid styles	44
34	Game vertical layout with grid styles	45
35	Horizontal grid layout with Flexbox layout for content divs content	46
36	Vertical grid layout with Flexbox layout for content divs content	46

Terminology

W3C

World Wide Web Consortium. The main international standards organization for World Wide Web.

HTML

HyperText Markup Language. The markup language used to build websites. The standard is maintained by the W3C.

HTML5

The latest version of the HTML standard. Is made up of multiple smaller specifications that are in varying stages of completion.

CSS

Cascading Style Sheets. It is a style sheet language that is used for describing

the look and feel of HTML documents. The standard is maintained by the W3C.

CSS3

The latest version of the CSS standard. Is made up of multiple smaller specifications that are in varying stages of completion.

JavaScript

JavaScript. A dynamic programming language. The de facto standard programming language client side web programming due to the fact that almost all browsers have a JavaScript interpreter. Standardized into ECMAScript that is maintained by Ecma International

Ecma International

An international standards organization for information and communication systems.

Ruby on Rails

A web application framework written in the Ruby programming language.

SQL

Structured Query Language. A programming language used in relational databases.

SQLite

An implementation of a SQL database written as a library in the C programming language. Small and lightweight and thus can be easily integrated in to other programs.

HTTP

Hypertext Transport Protocol. The protocol used for transporting web page content.

CRUD

Create Read Update Delete. A term describing the basic functionality of storing data.

jQuery

A very popular JavaScript library for simplifying browser scripting. Mainly helps working with the DOM.

Underscore.js

A JavaScript library that provides a big set of functional programming helpers to JavaScript without extending any of the built-in objects of JavaScript.

DOM

Document Object Model. It is a convention for representing and manipulating HTML using methods on its objects.

1 Introduction

The aim of the thesis was to research how to use the new and more effective technologies for developing HTML content provided by W3C that works on mobile devices for Qvantel Business Solutions Oy developers and the global developer community in general.

The research demonstrates the usage of new parts of the HTML5 and CSS3 specifications for creating content that can be viewed on both traditional desktop browsers and mobile devices effectively. The two new APIs were chosen from HTML5 with the goal of showing how to make more performant web pages for mobile devices. These two APIs are Web Workers that provide the ability to use the multiple CPU cores present in today's mobile devices and History interface for its ability to minimize the amount of roundtrips to the server serving the content and thus making the page function faster over possibly slower mobile connections while still maintaining the user experience that users are accustomed to.

The other parts are related to the ability to layout content on a web page differently based on the device viewing the page. This is achieved through the use of the two new layout modes that are part of the CSS3 specification called Flexbox layout and Grid layout. These together with Media Queries allow developers to have a single page that can serve multiple different devices effectively.

As a result of the research it should be clear how to move forward in web development when the aim is to also support mobile devices or any other device that is capable of showing web content. With the usage of mobile devices growing every day this should be of the utmost importance to anyone wishing to create new web content.

Objective

The objective is to show how to support multiple different devices using these new technologies. To achieve this the basic principles of the technology in question are discussed and after that a small example application is built using this technology. After reading this thesis it should be clear to any professional web developer what these four new technologies are, how to use them and when to use them.

2 Web Workers

Web Workers are a way to offload processing in to the background when working with JavaScript in the browser. Normally everything on a page is run in one single thread so blocking it while doing some resource intensive calculation can be detrimental to the user experience. There are two different kinds of Workers called Dedicated Worker and Shared Worker.

2.1 Concepts

When using Web Workers the main thread and the Worker communicate with each other by passing messages to each other using the onmessage event to listen to messages and the postMessage method to send the messages. The Worker itself runs in a different global context from DOM called DedicatedWorkerGlobalScope or SharedWorkerGlobalScore for dedicated and shared Workers respectively which both are separate from the current window scope. This allows the Worker to live even when the current window is closed or sharing it across multiple windows safely when using a shared Worker. To use dedicated or shared Workers the Worker or SharedWorker object is used to create the Worker.

A Worker may load other scripts or libraries using the importScripts function. A Worker can also create other works which are called Sub Workers. Workers themselves are somewhat heavyweight and it is not recommended to spawn a large amount of them due to the fact that the startup time and memory costs of a Worker are rather large.

2.2 Dedicated Worker example

As an example offloading of an expensive calculation to the background thread so that it does not block the DOM thread that is used. In this case the expensive calculation is to find all the prime numbers between two and ten million. Calculating this using JavaScript takes several seconds so running this in the DOM thread locks up the user interface for that period of time which is detrimental for the user experience.

```
// On the DOM thread
var primesWorker = new Worker("primes.js");
primesWorker.onmessage = function(event) {
    alert(event.data);
    primesWorker.terminate(); // terminate the Worker
};
primesWorker.postMessage(10000000);
```

```
// primes.js
// calculate prime numbers up to and including max
function primes(max) {
    var sieve = [], i, j, primes = [];
    for (i = 2; i <= max; ++i) {
        if (!sieve[i]) {
            primes.push(i);
            for (j = i << 1; j <= max; j += i) {
                sieve[j] = true;
            }
        }
    }
    return primes;
}

onmessage = function(event) {
    postMessage(primes(event.data))
};
```

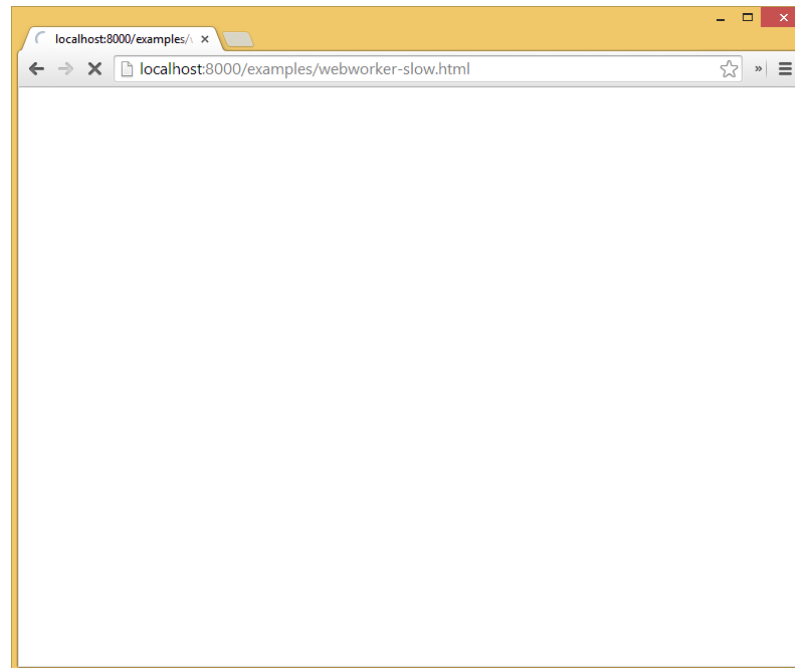


Figure 1: Calculating primes without a Web Worker. Does not even render HTML due to calculation being done inside head element

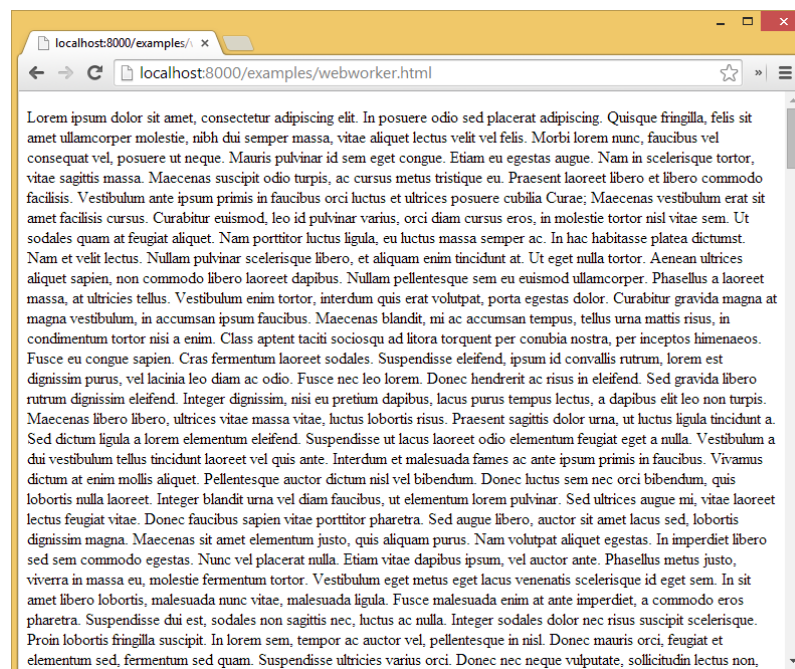


Figure 2: Calculating primes with a Web Worker

Running this on a web browser will after a short time show an alert box with the list of primes between two and ten million. During the time it takes to calculate this the page will continue to function normally (Figure 2). If Web Worker were not used to do this kind of a heavy calculation the page would become locked from the user while this calculation is ran (Figure 1). This is due to the single threaded nature of

web browsers.

2.3 Conclusions

In this section it was demonstrated how to offload calculations into another thread using Web Workers to achieve concurrency. The concurrency model of Web Workers is a very safe one due to the limitations of interaction between the main browser thread and the Worker thread. This helps to avoid many problems that are common with traditional multithreaded programming.

Some other use cases for Web Workers are image processing, compression/de-compression, web crawling and uploading large files to a server in the background. The file upload and web crawling use cases are interesting because the Web Workers exist out of band with the DOM thread the work would still continue even if the user changes the page and it does not exhaust one of the connections to the server which are limited to 2 in the HTTP 1.1 protocol.

The browser support for Web Workers is in general very good. The support of global client base is at 76% with the only major missing support being old Internet Explorer browsers (version 9 and below do not support Web Workers).

3 History interface

3.1 General

The HTML5 history interface allows manipulation of browser history through JavaScript. This can be used for all kinds of effects to improve the user experience.

One simple example of this would be to make a complex single page application using JavaScript while still preserving the back and forward functionality of the browser. Another example is to transform a complex multi-step form into a single page while still maintaining the same user experience. This is very useful with popular web frameworks like Ruby on Rails which prefer to use the CRUD flow where a single page load does a single action into the underlying database which becomes very problematic once the application starts to have complicated data insertion

processes with multiple steps. Using a multiple step form is how these kinds of data insertion processes are generally solved and when using the `pushState` method from the history interface it is possible to preserve the back and forward functionality that the users are used to even though in reality there is only one page load and submit.

3.2 Adding new history events

The `pushState` method takes three parameters. A state object, title and an optional URL.

State object The state object is a serializable JavaScript object that is associated with every new history entry created by the `pushState` method. When a user navigates to a history state a `popstate` event is fired and the state object is given as the `state` property of the event object.

Title Currently ignored by most browsers. It is safe to pass an empty string as the parameter for this. In theory it should change the title of the page in history but browsers use the title from the current page's headers.

URL An optional URL that the browser will change to. Note that the browser will not attempt to load this URL when changing the state but the browser may try to load the URL at some later point in time for example if the user chooses to reload the page.

Example:

```
history.pushState({ foo: "bar" }, "", "next-url");
```

Calling the `pushState` method is very similar to setting a new `window.location` to a URL fragment in the way that both will create a new history entry. Many helper libraries used to work with the History interface try to use URL fragments to replicate this behavior when `pushState` is not available but it is not always possible because `pushState` has some advantages compared to setting the `window.location`

- With `pushState` the new URL can be any URL with the same origin as the current URL. With `window.location` it is only possible to change the URL fragment. So, for example, with `pushState` it is possible to change the URL

form `foo.com/order/product` to `foo.com/contact` which is not possible with URL fragments. This makes the implementation of single page apps very hard with URL fragments.

- The developer is not forced to change the URL to create a new point in history. With `window.location` if the fragment is changed from `"#foo"` fragment to `"#foo"` fragment no new entry in browser history is created.
- It is possible to associate any kind of data with a new history entry with `pushState` into the state object. With changing the URL fragment the limitation is the maximum length of URL. Note that some browsers limit the size of the state object in History interface (for example Firefox has a maximum size of 640k, Internet Explorer 11 1MB and Chrome at least 10MB)

3.3 Changing current history state

The `replaceState` method has the same API as the `pushState()` method except it modifies the current entry in the history instead of creating a new one. This is useful when wanting the URL to respond to some user action on the page. For example a news site where the user is served with a long (possibly infinite) list of new news items as the user scrolls down. Changing the URL to point to the news item currently visible to the user makes it easier for the user to link to the news item if he/she wants to. But because new items are not added to the history if the user clicks the back button he/she would end up where he would expect he should. This is a functionality that cannot be replicated with URL fragments.

The main problem with the `replaceState` is that it is not possible to replicate its functionality with URL fragments like it is possible with `pushState`. This can limit the use cases for it if support for old browsers is required. For example Internet Explorer 9 does not support it. According to caniuse.com, the browser support for History interface is 73% of global web users.

3.4 Multistep form example using `pushState`

The goal is to make a simple multi-step order process where information about the order is collected from the in three distinct steps. The first step is selecting the

product being bought. The second step is collecting customers address information and the last step is the payment step.

This example is implemented with Ruby on Rails. The following schema is used in a SQLite database for orders

```
sqlite> .schema orders
CREATE TABLE "orders" (
  "id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
  "product_name" varchar(255),
  "price" varchar(255),
  "street" integer,
  "name" varchar(255),
  "zip" varchar(255),
  "payment_details" varchar(255),
  "created_at" datetime,
  "updated_at" datetime
);
```

The default form for creating a record like this could look something like this.

New order

Product name

Price

Street

Name

Zip

Payment details

[Back](#)

Figure 3: Order creation default form created by Ruby on Rails default form styles

This in itself is a fully functioning form for creating orders into the database, however what is really wanted is a step where the user first selects the device, then gives his information and as the final step gives the payment information.

As a multi-step form

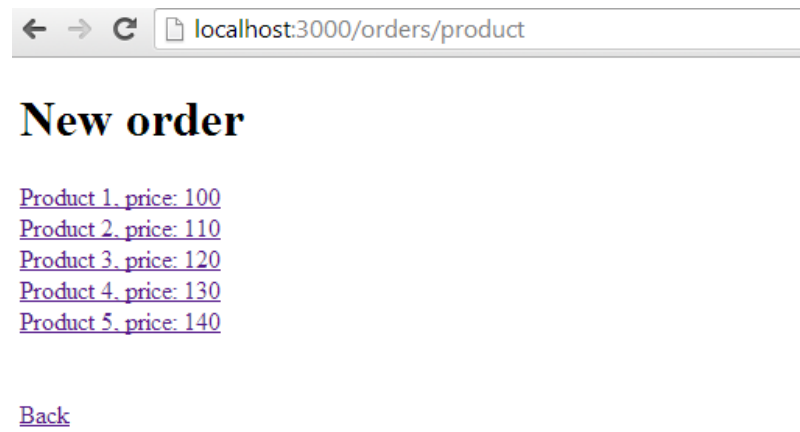


Figure 4: Product selection step

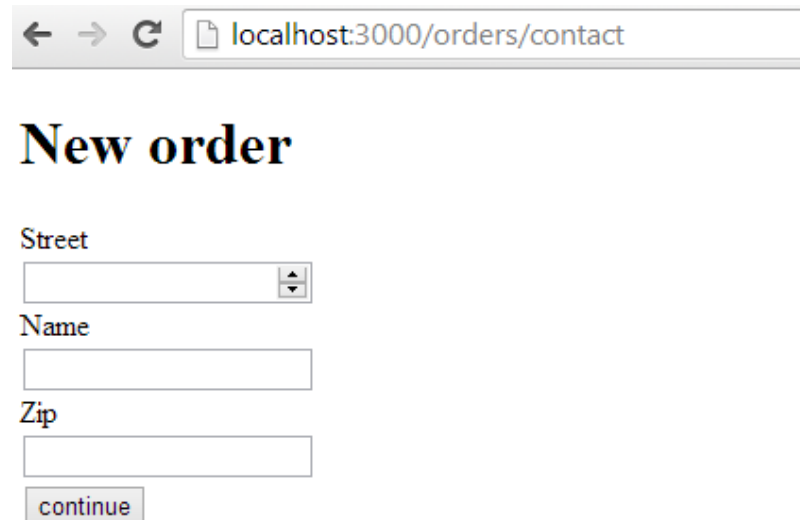
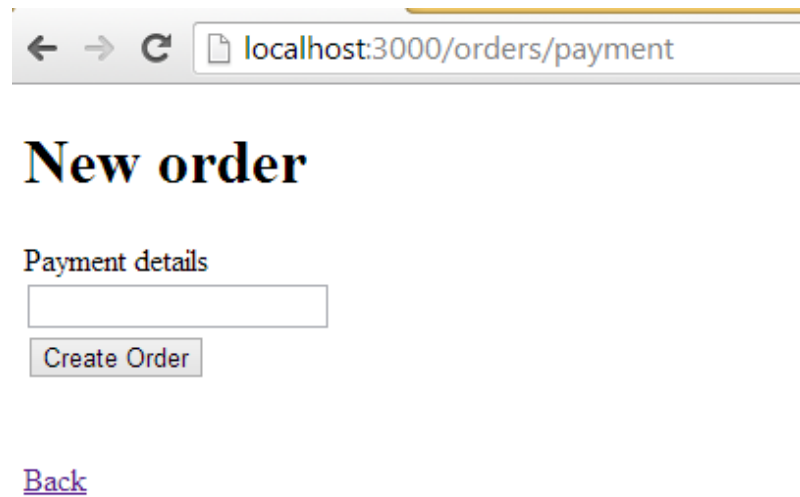


Figure 5: Contact information step



← → ↻ localhost:3000/orders/payment

New order

Payment details

Create Order

[Back](#)

Figure 6: Payment information and order saving

This way the user gets a better experience and still the model saving can occur as one single submit. Also, this way generates less requests to the server thus generating less load. If validation between the steps is wanted that can be easily handled with JavaScript using AJAX.

This works by adding a small piece of JavaScript code that creates new state or replacing the current one in browser history. For creating new events this is done through the `pushState` method and replacing of current state is done through the `replaceState` method from the history object.

3.5 Example of updating current session state

Here is a simple example of a web page with multiple articles. Once user scrolls down enough that the previous article is no longer visible the URL will change into the next article's URL. If the user presses the back button on the browser the user will be taken to the page he was in before navigating to this page. If from that page the user presses forward he will end up going to the URL of the article he was reading before.

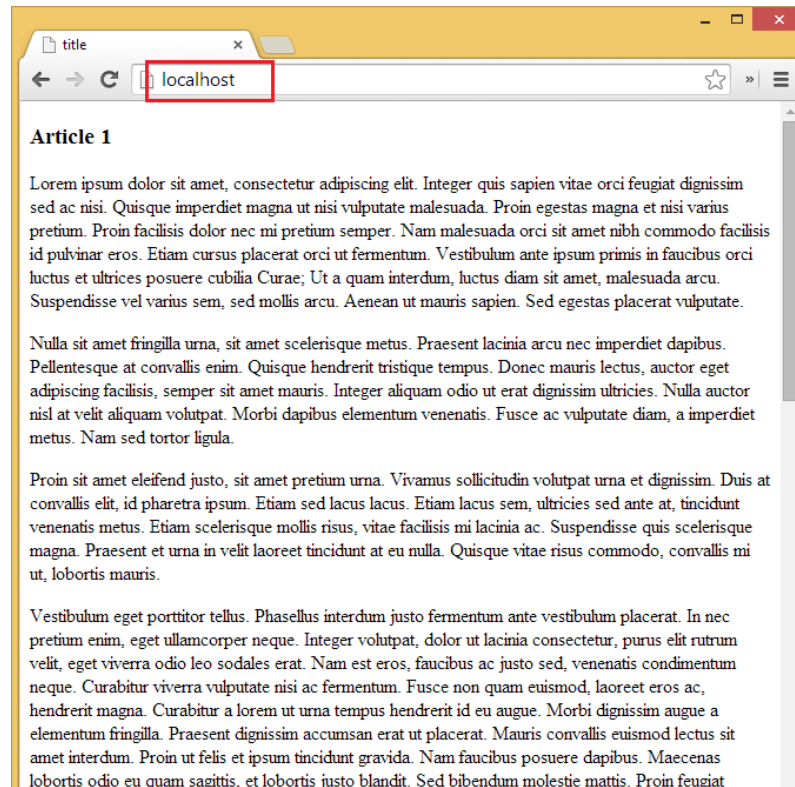


Figure 7: User comes to web page

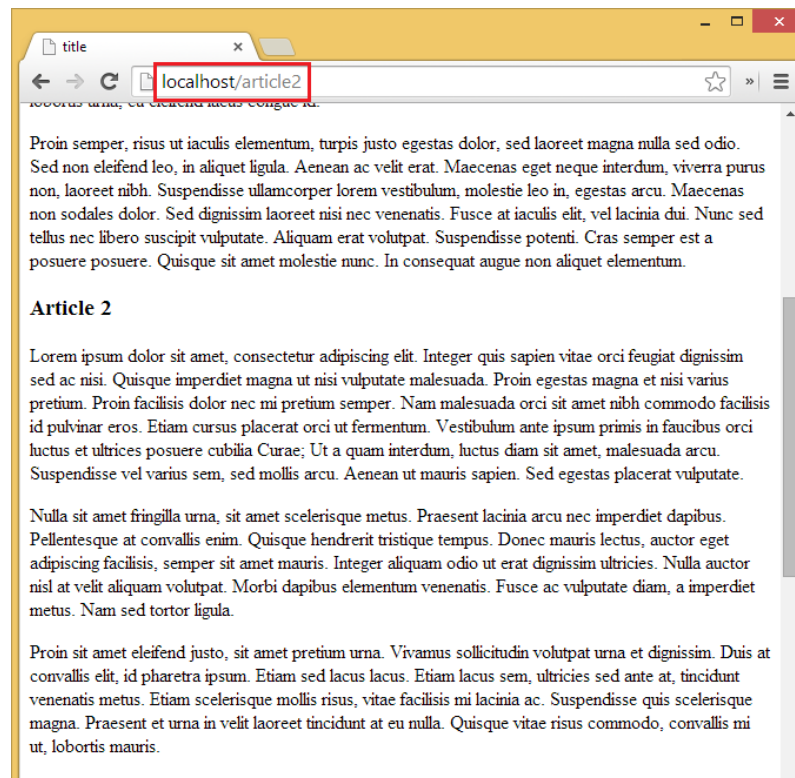


Figure 8: User scrolls down to Article 2

As can be observed once the user scrolls down to article 2 the URL is changed to <http://localhost/article2>. In this case if the user came from <http://google.com> if he

were to press the back button he would be taken there. And if he chooses to press forward on that page he would be taken to <http://localhost/article2>

Implementing this functionality itself is very simple. For this example the articles are located inside the article element with the id attribute being the wanted URL part. The complete code for this example is here with only using jQuery for handling DOM and Underscore.js for throttling the actually changes to the URL only happening once every 100 milliseconds.

```
// save jQuery window object so we don't
// have to recreate all the time.
var $window = $(window);

// get the current window top and bottom and
// wanted elements top and bottom in px and do
// comparison to find out if the element is in view.
function isInView($el) {
    var top = $window.scrollTop(),
        bottom = top + $window.height(),
        elTop = $el.offset().top,
        elBottom = elTop + $el.height();

    return elTop >= top
}

// make a throttlet function so we don't
// cause so much load on the cpu
var throttled = _.throttle(function() {
    var last;
    $('article').each(function() {
        var $this = $(this);

        if(isInView($this)) {
// if element is in view save it to last variable
            last = $this;
// and break loop
            return false;
        }
    });

// get the id from the element
    var id = last.attr('id');
```

```
// set the state
history.replaceState({ article: id }, id, id);
} , 100);

// call our thorttlet function on every scroll event
$(window).scroll(throttled);
```

3.6 Conclusions

The main problems with the History interface is if support for old browsers is required that do not implement the History interface. There are libraries that mitigate this problem quite well by using URL fragments, however because it is not possible to replicate all the functionalities of the History interface with them they do not always work. The other problem is if the user presses refresh. To fix this the server should respond to the URLs created by the History interface and the developer has to somehow the information filled in the previous steps the client side. For storing the state usually the state object is good enough but sometimes resorting to Web Storage and Indexed DB may be necessary if the state would be too big to save into the state object.

The problem with supporting older browsers is no longer a big problem. The latest versions of all major browsers except Opera Mini support the History interface which gives the History interface a very good 78% support of users according to caniuse.com

4 Flexbox layout

The main idea behind Flexbox is to provide a way to arrange elements in a predictable way when viewing the content on different screen size and display devices. In Flexbox layout there are elements that are placed inside a container. These elements are then laid out in lines inside the container in the way specified with CSS properties on the container and the items.

4.1 Flexbox concepts

Flex container The element containing flex items. An element is made into a flex container by giving it the flex value to the display property.

Flex line A line of flex items. This isn't a HTML element but is used as a concept for the layout logic.

Flex item An element inside the flex container.

main-size and cross-size The width or height of a flex container. This depends on the direction of the Flexbox layout.

main-axis and cross-axis The main-axis is the direction in which elements flex items are laid out inside the container. Cross-axis is perpendicular to the main axis.

main-start Located at the beginning of the main-axis. Items inside the container are placed starting from it towards the main-end.

cross-start Located at the beginning of the cross-axis. Flex lines are placed inside the container starting from it towards the cross-end.

main-end and cross-end End of the container in main-axis and cross-axis directions.

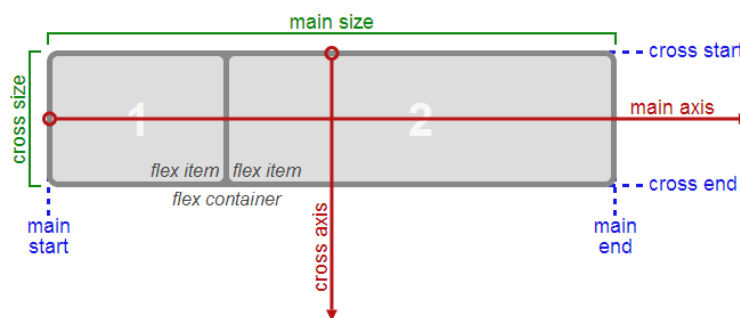


Figure 9: Illustration of the Flexbox concepts when the direction is 'row' (originally from CSS Flexible Box Layout Module Level 1 W3C Working Draft)

Flexbox layout can be laid in four different directions which in turn change the direction of the axes. These directions also take into account the current writing mode. What this means is that for example row direction with a western writing mode places the flex lines from top to bottom with the main axis direction going

from left to right, however then when using a different writing mode like Japanese the main axis would go from top to bottom and the lines would be laid from right to left.

row Main-axis is in the same direction as the current writing modes inline axis.

Main-start and main-end directions are the same as the start and end directions in the current writing mode.

row-reverse Same as 'row' with the main-start and main-end directions swapped.

column Main-axis is in the same directions as the current writing modes block axis. Main-start and main-end directions are the same as the block-start and block-end directions of the current writing mode.

column-reverse Same as 'column' with the main-start and main-end directions swapped.

4.2 Basic layout and wrapping

In this section the basic layout and wrapping in a Flexbox layout is demonstrated using an example. The following basic HTML and CSS is used for the example, which will result in page as shown in Figure 10

```
<div class="Flexbox">
  <div>red</div>
  <div>green</div>
  <div>blue</div>
</div>
```

```
/* colors for the elements so we can see them grow/shrink */
div > :nth-child(1)
{
  background: red;
}
div > :nth-child(2)
{
  background: green;
}
div > :nth-child(3) {
  background: blue;
```



```
}  
  
.Flexbox  
{  
  /* basic styles */  
  width: 100%;  
  height: 150px;  
  font-size: 24px;  
}
```

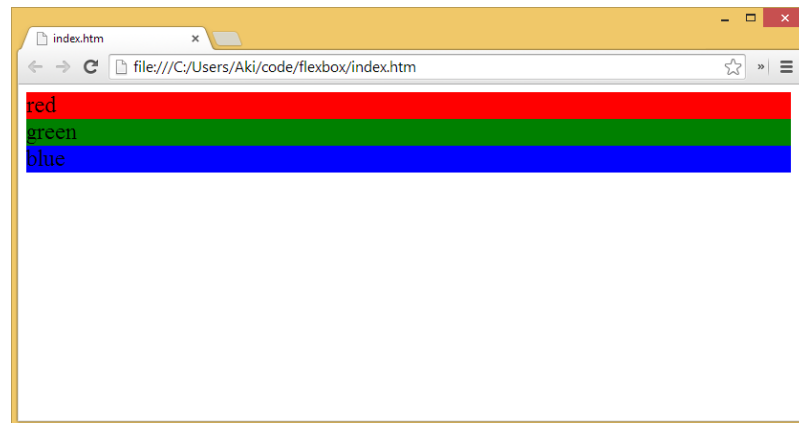


Figure 10: Without Flexbox

To make an element use Flexbox layout the developer first has to specify the display mode as flex and give the flex direction. Here is a CSS snippet setting up a Flexbox container with flex direction to horizontal (rows) to elements with the class Flexbox. This will make the flex items inside the flex container be laid out in rows (Figure 11)

```
.Flexbox {  
  display: flex;  
  flex-direction: row;  
}
```

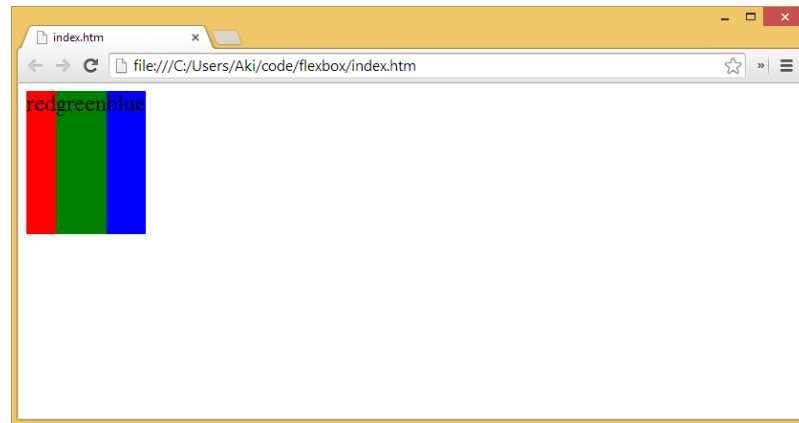


Figure 11: With simple Flexbox layout

It is possible to control how the space inside the flex container is used by the flex items using the following CSS properties:

flex-grow Used to define the flex grow factor that tells how an element inside a Flexbox container will grow relative to the other elements inside the container. By default set to 1.

flex-shrink Used to define the flex shrink factor. Works the same way as flex-grow except for shrinking the elements. By default set to 1.

flex-basis Used to define the default size of an element before the free space is distributed according to the flex grow and flex shrink factors. Takes the same values as the CSS width property. If set to auto uses the elements width or height property depending on the direction of the Flexbox layout.

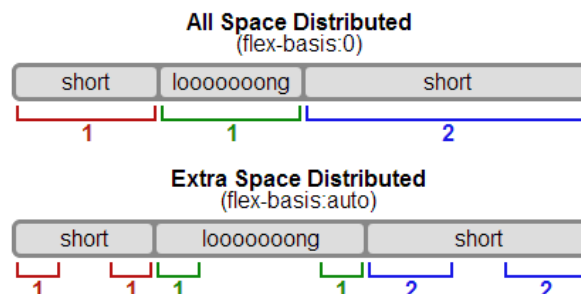


Figure 12: Illustration of how flex-basis. Red and blue numbers are flex-grow factors. (originally from CSS Flexible Box Layout Module Level 1 W3C Working Draft)

Now it is possible to give the Flexbox a simple directive to automatically fill the container with its content elements (Figure 13). Here is a CSS snippet for that:

```
.Flexbox > div
{
  flex-grow: 1;
  flex-shrink: 1;
  flex-basis: auto;
}
```



Figure 13: Flexbox automatically filling the container with its content

It is also possible to control how big of a proportion of the available space each flex item inside the flex container takes by using the flex-grow CSS property. (Figure 14.)

```
div > :nth-child(2)
{
  flex-grow: 10;
}
```



Figure 14: With green element having bigger grow factor

This in itself is not something that cannot be done with percentage based layout and floats. Next, the wrapping of the flex items inside the flex container to multiple rows is demonstrated. This is something which is not easily replicated using the block

model. First, the developer gives the flex items inside the flex container a minimum width or actually have enough content inside the elements to fill the space and set the flex-wrap property to wrap which is done by adding the following CSS rules:

```
.Flexbox > div
{
  min-width: 200px;
}

.Flexbox
{
  flex-wrap: wrap;
}
```

With this if the screen width is less than 600px but more than 400px the user will see one row with 2 elements and a second one with 1 element (Figure 15). If the screen is less than 400px wide all the elements will wrap into their own rows (Figure 16). The flex-grow property is also taken into account when wrapping to multiple flex lines happens (Figure 17). This is one of the features that cannot be replicated by just using percentage based widths and floating elements because when floats there is no way to tell the items on lines with less items to use more of the available space.

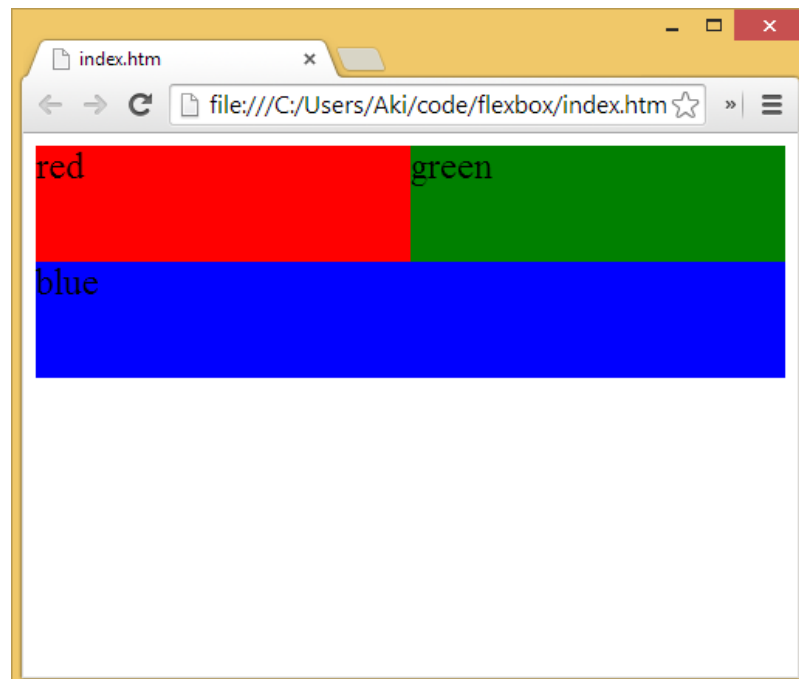


Figure 15: With 500px wide window

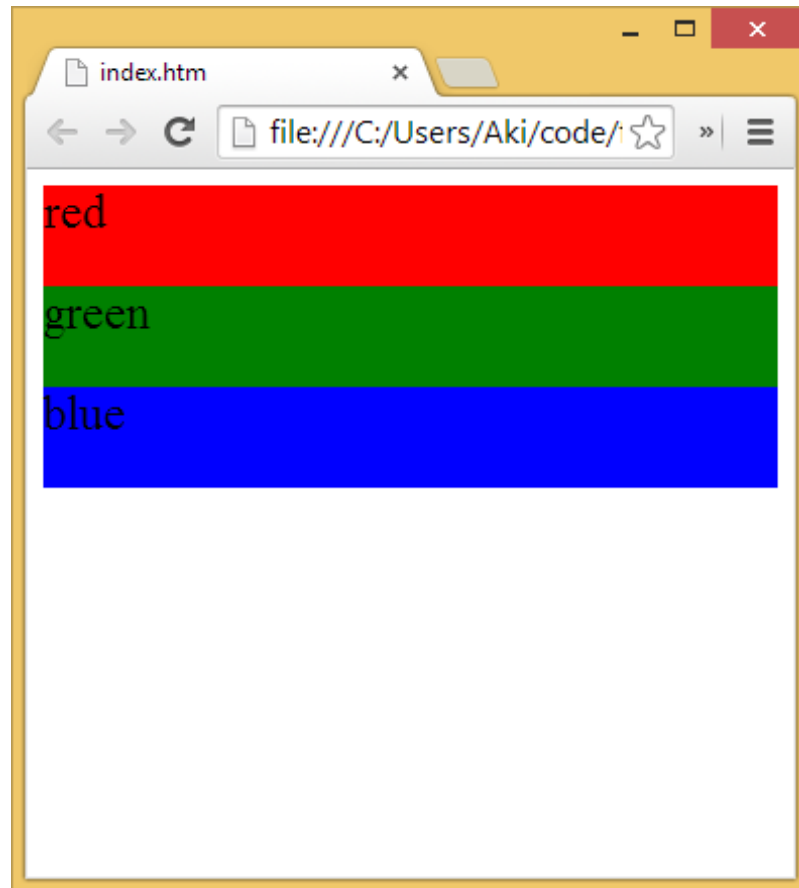


Figure 16: With 380px wide window

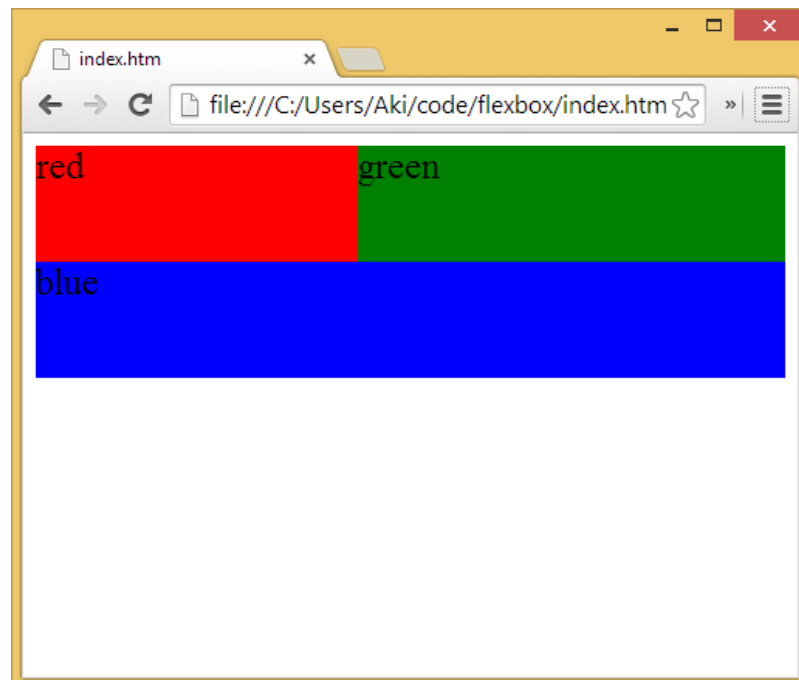


Figure 17: With 500px wide window. Wrapping with green element having bigger grow factor

The wrapping behavior of Flexbox layout cannot be replicated using the block layout mode without the use of very complicated Media Queries and JavaScript code. Some very simple use cases could be replicated but when more items inside the container are added it gets more and more difficult to get the same layout without using Flexbox layout. Also, in general having a native way to do the layout in the wanted way without relying on JavaScript is almost always faster to render.

4.3 Ordering

Flexbox also allows for easy control of ordering of elements inside the container. By default the elements are in the order they are in the source but this can be controlled with the `order` CSS property on the flex items. For example continuing on the previous example it is possible to make the blue element appear first in the list (Figure 18) with the following CSS. It should be noted that when using the `order` property the ordering number for all elements must be given.

```
div > :nth-child(1)
{
  order: 2;
}
div > :nth-child(2)
{
  order: 3;
}
div > :nth-child(3)
{
  order: 1;
}
```

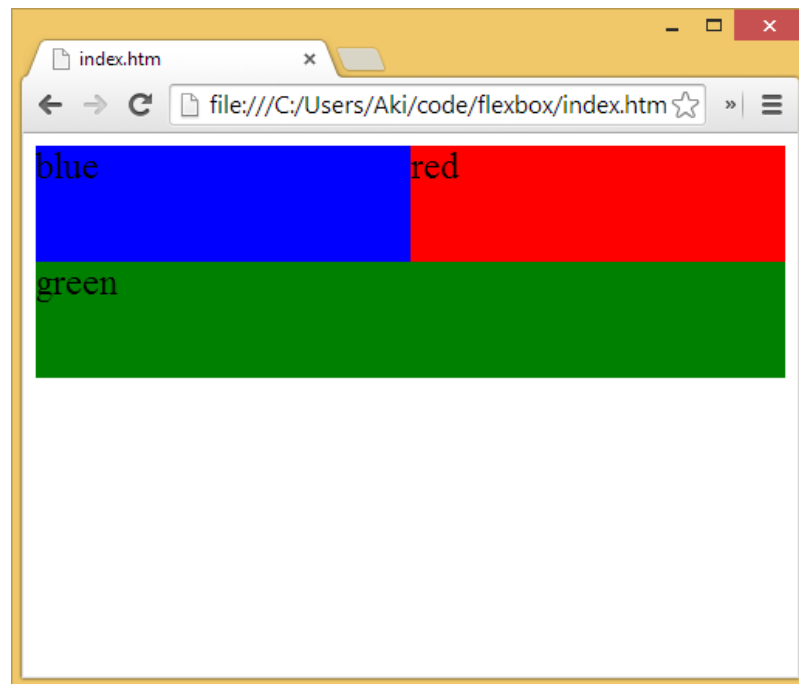


Figure 18: With the blue third (blue) element ordered to be first

4.4 Alignment

With Flexbox layout the developer is not forced to fill the whole space with content. Instead the developer is given with multiple ways of leaving margins around the flex items and alignment of them.

4.4.1 Alignment with auto margins

Auto margins in a Flexbox layout are enabled by setting the `margin-left` or `margin-right` CSS properties to `right` or `left`. The effect itself is very similar to using auto margins in block layout in that it can be used to push elements to either side of main-axis. One of the most common uses for auto margins is to split items in a menu bar into two distinct groups. In this example there is a navigation bar (Figure 19) which is built using the following HTML and CSS.

```
<nav>
  <ul>
    <li><a href="/home">Home</a></li>
    <li><a href="/forum">Forum</a></li>
    <li><a href="/help">Help</a></li>
    <li><a href="/login">Login</a></li>
  </ul>
```

```

</nav>

nav > ul
{
  display: flex;
  list-style-type: none;
  border-radius: .2em;
  padding: .2em .5em;
  background-color: gray;
}

nav > ul > li
{
  /* keeps the items from getting too small for the content in
  them*/
  min-width: min-content;
  border-radius: .3em;
  padding: 0 .25em;
}

```



A horizontal navigation bar with a gray background and rounded corners. It contains four blue text links: "Home", "Form", "Help", and "Login", all in a sans-serif font.

Figure 19: Navigation bar using Flexbox layout

Now to split the help and login items on the navigation bar, only the addition of one single CSS property on the correct flex item is required. (Figure 20)

```

nav > ul > :nth-child(3) {
  margin-left: auto;
}

```



A horizontal navigation bar with a gray background and rounded corners. It contains four blue text links: "Home", "Form", "Help", and "Login". The "Help" and "Login" links are separated from the others by a significant margin on the right side.

Figure 20: Navigation bar with auto margin

4.4.2 Main axis alignment

Alignment over the main axis is implemented with the justify-content property. It is normally used when flex items inside a container cannot grow or shrink or they can grow but have reached their maximum size. Justify-content property also gives

some control over the alignment when items overflow a line. There are five different justify-content modes which are as follows:

flex-start Items are packed to the beginning of the line.

flex-end Items are packed to the end of the line.

center Items are centered on the line.

space-between Evenly split the available space between the items but not between the items and container.

space-around Split the available space between the items and between the items and container at the beginning and end of the line. The space between the container and the first/last item is half the space that is left between the items them self.

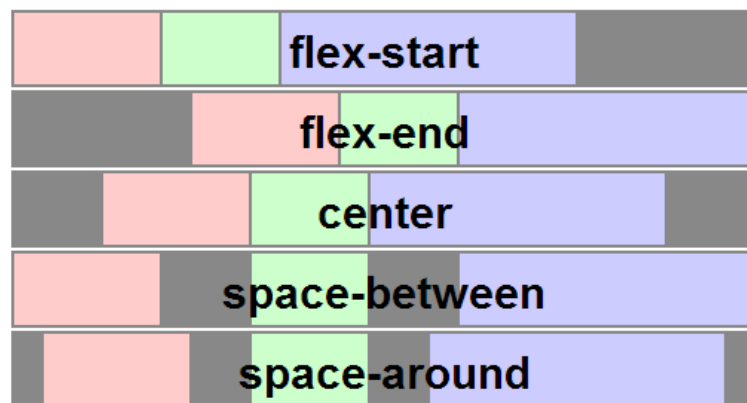


Figure 21: Illustration of the different main axis alignment modes. (originally from CSS Flexible Box Layout Module Level 1 W3C Working Draft)

4.4.3 Cross axis alignment

With Flexbox there is also the possibility of aligning in the perpendicular direction of the main axis (cross axis) in a very similar way as with justify-content. When aligning on cross axis there are two options. One is for aligning all the flex items inside the container using the align-items property on the flex container and single items using the align-self property on a flex item. The same five alignment options are available for both align-items and align-self which are as follows

flex-start Pushes the items cross-start margin to the cross-start of the line. For example, if using vertical Flexbox flow all the items would have their top margin against the top of the container. Overflows over the cross-end of the line if container is too small.

flex-end Pushes the items cross-end margin to the cross-end of the line. Overflows over the cross-start of the line if container is too small. In essence it is the opposite of flex-start.

center Centers the items on the line. Overflows in both directions if the container is too small for the item.

stretch Stretches the items cross-start and cross-end to the containers cross-start and cross-end. Will respect the constraints from min-height, min-width, max-height, and max-width properties of the item. If content of the item does not fit the content will overflow from the item.

baseline Tries to put the content of the items in the same line.

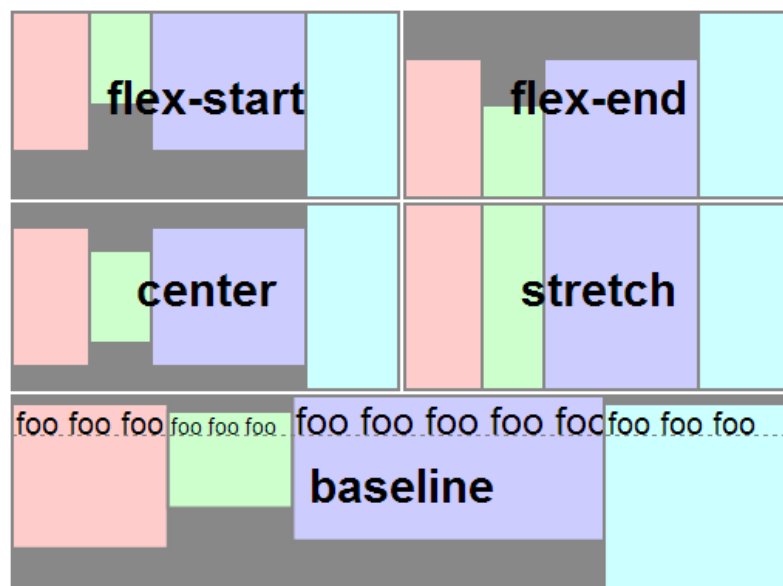


Figure 22: Illustration of the different cross axis alignment options. (originally from CSS Flexible Box Layout Module Level 1 W3C Working Draft)

4.4.4 Alignment of item lines

With Flexbox layout it is possible to align lines inside the flex container in respect to each other. This is done using the align-content property that has six different possible values as follows.

flex-start Places the lines starting from cross-start edge of the container leaving the rest of the container empty

flex-end Places the lines starting from the cross-end edge of the container leaving the rest of the container empty.

center Centers the lines inside the container. Leaving equal amount between the first item and cross-start and last item and cross-end

stretch Fills the empty space in the container by stretching the lines. If there is no space left acts like flex-start

space-between Fills the empty space by leaving empty equal space between the lines.

space-around Fills the empty space by leaving space between the lines and between the first and last line and edges of the container. The space between the edges of the container and the lines is half of what used for between items.

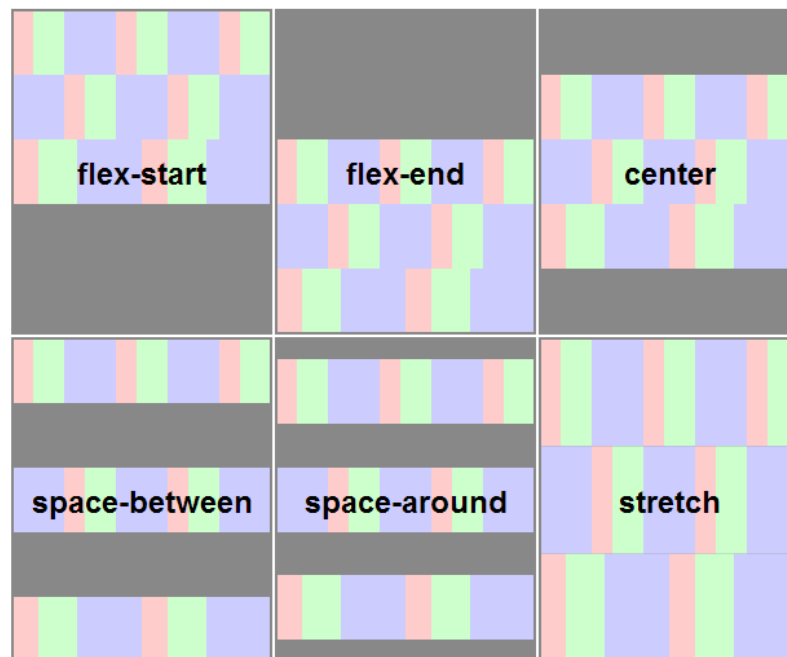


Figure 23: Illustration of the alignment between lines. (originally from CSS Flexible Box Layout Module Level 1 W3C Working Draft)

4.4.5 Example

As an example a simple responsive website layout is implemented. In traditional desktop size browser windows the nav, content and aside sections should be laid out

next to each other. When going from a big desktop size browser window to a small mobile browser window the nav, content and aside sections should be stacked on top of each other (Figure 24). A very simple HTML content with a header, content, nav, aside and footer sections will be used for the website. In this HTML the content, nav and aside sections will be located inside a Flexbox container that the Flexbox layout will be applied to. (Figure 25)

```
<header>Header</header>
<div class="Flexbox"> <!-- the flex container -->
  <article>Content</article>
  <nav>Nav</nav>
  <aside>Aside</aside>
</div>
<footer>Footer</footer>
```

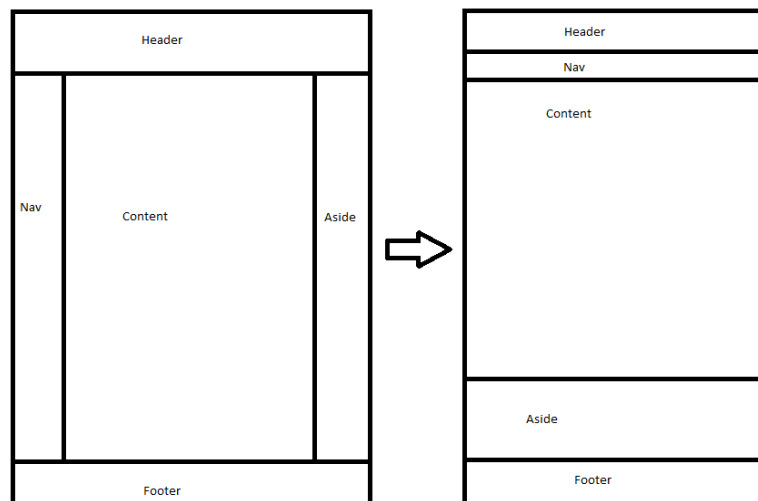


Figure 24: Illustration of the transition of the layout from large screen to small screen

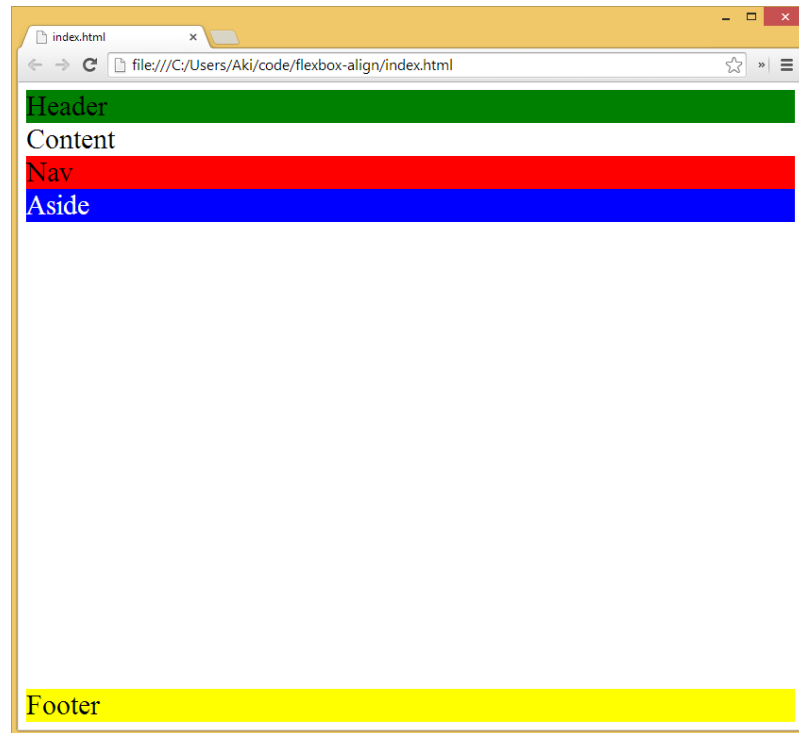


Figure 25: Layout before Flexbox layout

As observed it can be seen that the page looks almost correct already for the small screen devices except for the fact the nav, aside and content are in the wrong order. Flexbox layout will now be applied to the flex container that will make the site look correct on large screens. This is achieved by setting the correct display and flex properties to the flex container in the style sheet. After that it is possible to set the appropriate order, grow, shrink and basis values for the nav, article and aside elements to achieve the look that was wanted (Figure 26)

```
.Flexbox {
  min-height: 600px;

  display: flex;
  flex-direction: row;
  flex-wrap: wrap;
}

.Flexbox > nav
{
  background: red;
  flex-grow: 1;
  flex-shrink: 5;
  flex-basis: 20%;
}
```

```
    order: 1;
}

.Flexbox > article
{
    background: white;

    flex-grow: 3;
    flex-shrink: 1;
    flex-basis: 60%;

    order: 2;
}

.Flexbox > aside
{
    background: blue;
    color: white;

    flex-grow: 1;
    flex-shrink: 5;
    flex-basis: 20%;

    order: 3;
}
```

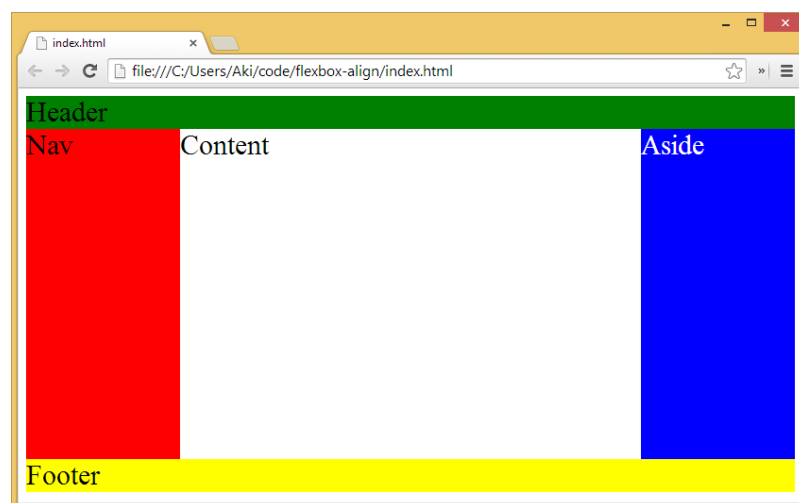


Figure 26: Flexbox layout on a wide screen

Now that the large screen use case is implemented the problem is that if it is viewed on a small device the middle section of the page will get squashed into a very

narrow element. To fix this CSS3 Media Queries are used to apply the appropriate CSS rules when the screen size gets small enough.

```
/* apply these rules when screen width is less then 500px */
@media all and (max-width: 500px) {
/* change Flexbox direction from horizontal to vertical */
  .Flexbox {
    flex-wrap: column;
    flex-direction: column;
  }

/* Change the order of Flexbox items*/
  .Flexbox > nav
  {
    order: 1;
  }

  .Flexbox > article
  {
    order: 2;
  }

  .Flexbox > aside
  {
    order: 3;
  }

  .Flexbox > nav, .Flexbox > aside {
    min-height: 35px;
    max-height: 35px;
  }
}
```



Figure 27: The layout of the example on a narrow screen

As can be observed from Figure 27 the page now shows on small devices as was wanted. One great advantage of the Flexbox layout is that it is just another CSS display mode that can be applied to only the parts of the page where it is needed. In the example the header and footer elements do not use Flexbox layout at all, which means that it is possible to take Flexbox layout to use into existing pages by applying it only to the parts it is needed for.

4.5 Conclusions

The main problem with the Flexbox layout for now is browser support because the spec is still in the candidate recommendation phase of the W3 standardization process. According to caniuse.com full support for the latest version of the specification is at 54% while partial is at 26%. This leaves between 46 to 20% of the users in the world without being able to correctly view a page using Flexbox layout which can be a big problem in some markets. With no easy way to replicate the same functionality this can be a stopper in taking Flexbox layout into use. For example if developing for a corporate customer that still uses an old version of Internet Explorer or emerging markets with low powered smartphones that don't use the latest browsers.

Still using Flexbox for handling some parts of your web page layout is highly recommended. While it is possible to replicate some of the functionality of Flexbox layout using media queries and floats relying complicated CSS frameworks that are made of hundreds or thousands of lines of instructions and HTML content that is specifically made for that framework is required. Also the big CSS frameworks require that they are used everywhere on your whole page while Flexbox can be used only for the parts it is needed for makes taking them into use for existing projects very complicated.

5 Grid layout

5.1 General

Grid layout is another new layout mode coming with CSS3. It is heavier weight than Flexbox layout and is meant for handling the layout of complex online applications. Grid layout allows the content to be laid in a very different way depending on the device being used to view the page. Not only can the size and proportions of the elements be changed so can the ordering of them. In a way grid layout is the first time that CSS is getting close to its old promise of detaching the layout from the content. It is powerful enough to truly create almost any kind of a layout from just a group of elements inside a container (where the container can be the body element of the page).

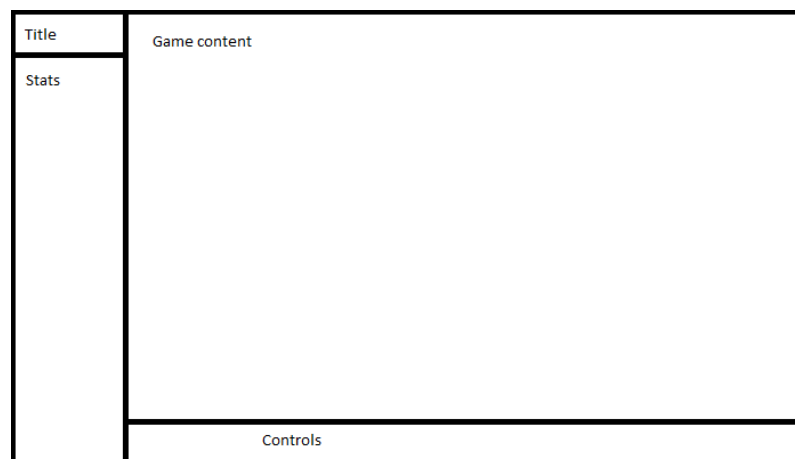


Figure 28: Horizontal game layout



Figure 29: Vertical game layout

Both of these layouts should use the same HTML and CSS with Media Queries handling the changing of the Grid layout from one form to another.

5.2 Concepts

In some ways Grid layout works in a very similar way to tables, however, it has the advantage of not being tied to the content structure which allows the layout to change depending on the screen size, orientation and aspect ratio. It is also possible to combine cells with grid layout. For example one element could be the first three rows of column one. This is similar to how it is possible to merge columns when using tables but with grid layout it is possible to do this in both horizontal and vertical direction without touching the HTML content.

This should be very familiar to anyone who has worked with HTML before the rise of CSS when a lot of styling on a web page was done using tables. But these table based layouts were very brittle to different screen sizes and wasted a lot of space.

As an alternative many pages were created with fixed layout that did not react at all to the browser's window size. Grid layout was made to address these problems. It gives the developer tools to divide the available space into columns and rows which are then easy to fill with wanted content. The position of each element in Grid layout can be changed with only CSS changes so it is possible to have the layout respond to changes in the screen size just by using CSS3 Media Queries.

To make an element use grid layout inside of it developers give its display property the value grid. Along with it, developer indicates how many columns and rows developer wants in the grid. Developer controls these with the grid-columns and grid-rows properties. Both these properties take a space separated list of how wide and high columns or rows are wanted. For example, giving the value '100px 100px' to grid-columns means that the developer wants two 100 pixel wide columns.

After setting up the grid the grid-column and grid-row properties are defined on the elements inside the container. They take a simple running number of the column or row the element should take. Grid-column-span or grid-row-span properties can also be given which tell the element to span into multiple columns or rows. The default value for both grid-column-span and grid-row-span is '1' so if the element is wanted to spawn two rows the grid-row-span property is the given value '2'.

5.3 Example

As an example the horizontal and vertical game layouts showed earlier are implemented. For this there is a very simple base HTML with some basic styles (Figure 30). It should be noted that for all grid layout related CSS the "-ms-" prefix is used due to the fact that Microsoft with its Internet Explorer browsers is the only vendor that has implemented the grid layout.

To implement the example the two wanted layouts (Figures 28 and 29) are observed for ways to split them into columns and rows. It can be easily seen that it is possible to split the content into grids of two columns and three rows in both cases (Figures 31 and 32).

```
<div id="grid">  
  <div id="title">Title</div>  
  <div id="stats">Stats</div>
```

```
<div id="content">Game content</div>  
<div id="controls">Controls</div>  
</div>
```

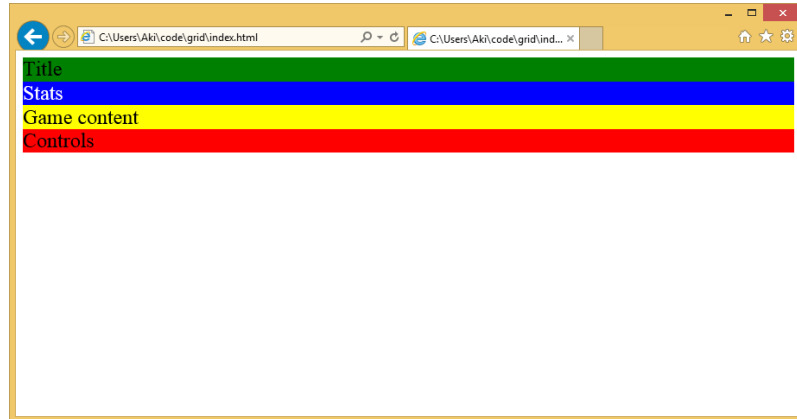


Figure 30: Basic layout with simple styles

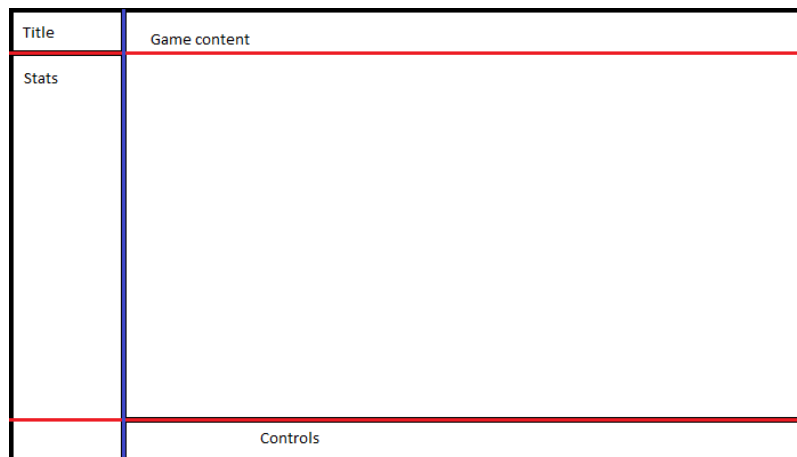


Figure 31: Horizontal layout with red lines showing row splits and blue lines showing column split

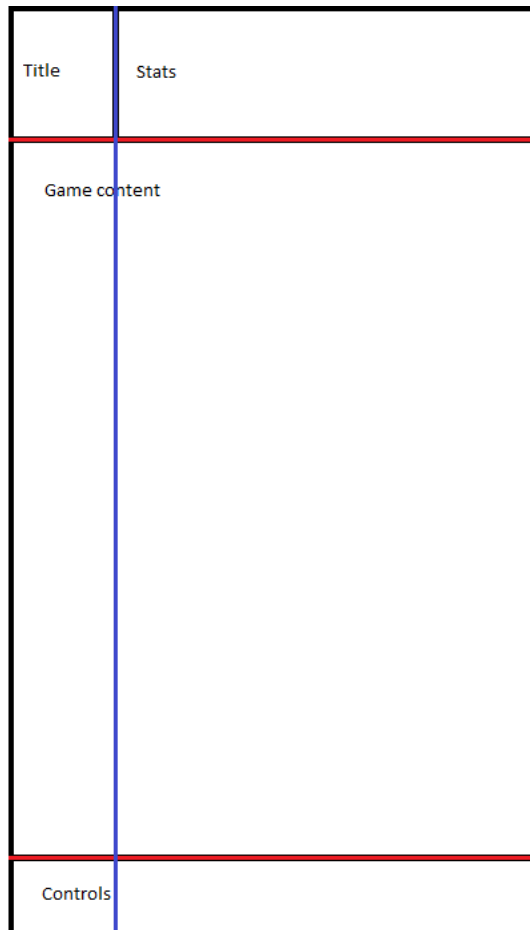


Figure 32: Vertical layout with red lines showing row splits and blue lines showing column split

The display mode can be set now for all viewing modes as `grid` and the grid set up with two columns where the first one is 100px wide and the other using as much space as needed for its content. The three rows can now be set up with the first and the third row being 100px high and the second row using as much space as needed for its content.

```
#grid {
  display: -ms-grid;
  -ms-grid-columns: 100px 1fr;
  -ms-grid-rows: 100px 1fr 100px;
}
```

Now that the grid layout set is out the grid content needs to be specified in horizontal (landscape) mode using a media query. In the horizontal layout the first row is set up to contain the title and content, the second row with stats and content and the third row with stats and controls.

```
@media (orientation: landscape) {
```

```

/* title only takes first rows first column */
#title
{
    -ms-grid-column: 1;
    -ms-grid-row: 1;
}

/* stats starts at second row first
column but spans over 2 rows */
#stats
{
    -ms-grid-column: 1;
    -ms-grid-row: 2;
    -ms-grid-row-span: 2; /* stats to span to third row */
}

/* content starts from first rows
second column but spans 2 rows */
#content
{
    -ms-grid-column: 2;
    -ms-grid-row: 1;
    -ms-grid-row-span: 2; /* content span into second row */
}

/* controls only takes the third rows second column */
#controls
{
    -ms-grid-column: 2;
    -ms-grid-row: 3;
}
}

```

Now that the CSS has been applied, when the page is in horizontal mode it can be observed that the grid layout is working correctly when using a browser that is in landscape orientation. (Figure 33)

With the horizontal layout working, CSS styles for the vertical layout can be added. This is done by adding the styles in a similar fashion as before but instead a media query is used where the orientation is portrait instead of landscape and the grid setup is modified for the vertical layout.



Figure 33: Game horizontal layout with grid styles

```

@media (orientation: portrait) {
  /* title only takes first rows first column */
  #title
  {
    -ms-grid-column: 1;
    -ms-grid-row: 1;
  }

  /* stats only takes first rows second column */
  #stats
  {
    -ms-grid-column: 2;
    -ms-grid-row: 1;
  }

  /* content takes second rows both columns */
  #content
  {
    -ms-grid-column: 1;
    -ms-grid-row: 2;
    -ms-grid-column-span: 2; /* content span both columns */
  }

  /* controls takes thord rows both columns */
  #controls
  {
    -ms-grid-column: 1;
    -ms-grid-row: 3;
    -ms-grid-column-span: 2; /* controls spawn both columns */
  }
}

```

```
}  
}
```

After applying this CSS it can be seen that the content layout changes when using a narrow (vertical) view the page (Figure 34). It should also be remembered that because of the grid layout itself is just another display mode in HTML the items inside the grid layout can use a different layout mode for its content (Figures 35 and 36)



Figure 34: Game vertical layout with grid styles

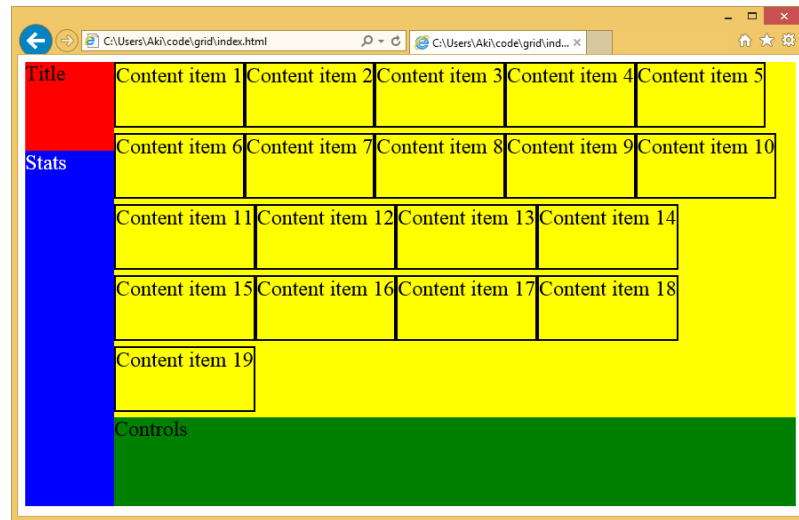


Figure 35: Horizontal grid layout with Flexbox layout for content divs content

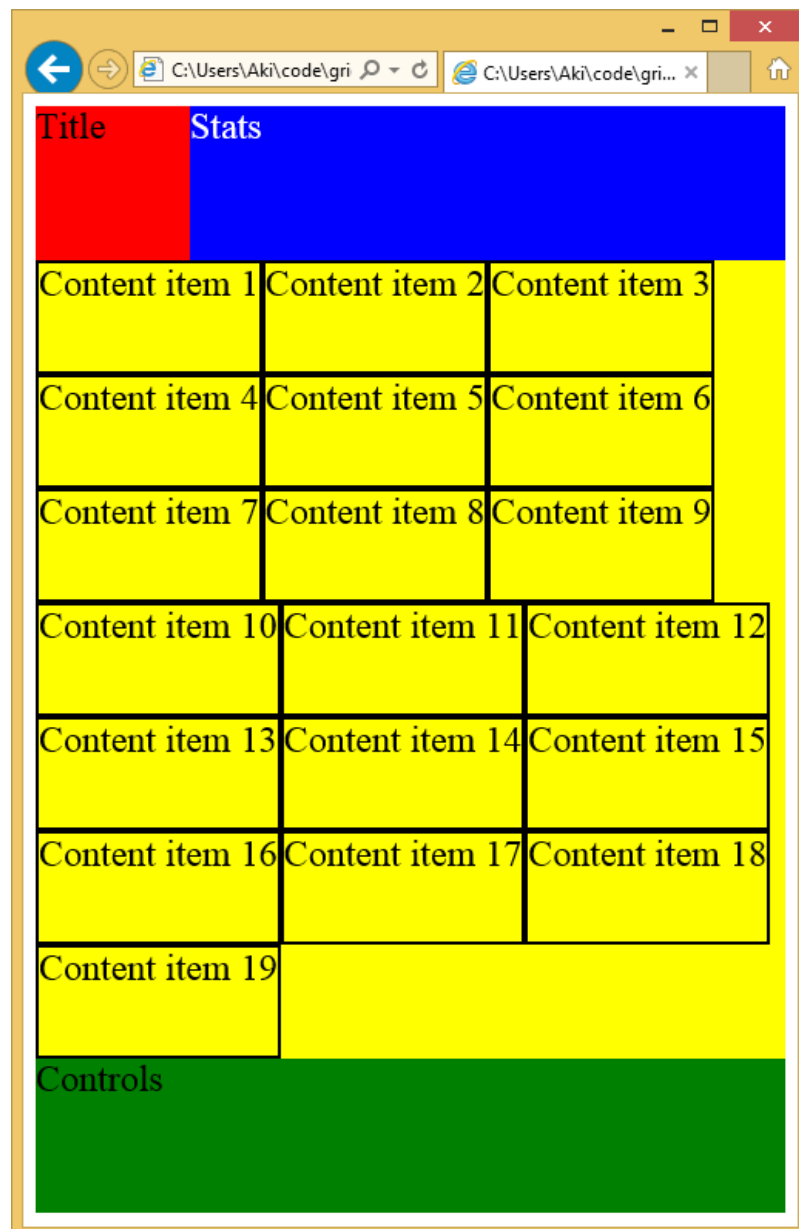


Figure 36: Vertical grid layout with Flexbox layout for content divs content

5.4 Conclusions

The major problem with adopting the grid layout is that browser support for it is limited to the versions 10 and 11 of Internet Explorer. According to caniuse.com, only 10% of the global market share currently support grid layout. The only use case for now where the use of grid layout could be recommended is building custom software for a specific client's private use where browser version can be controlled. But as a future technology grid layout can be seen as very promising due to its ability to greatly simplify CSS styles and help truly separate the HTML content from the CSS styles. Most of the features provided by grid layout can be replicated by using popular CSS grid frameworks that split the page into n amount of columns; these force developers to have their content contain the rows, which makes it very difficult to have different layouts for different devices using Media Queries.

6 Summary

The aim of the thesis was to explore some of the new HTML5 and CSS3 technologies and find out how they could be used to provide a better experience to mobile users who have screen sizes and aspect ratios very different from traditional desktop users. At the beginning, two different ways to make the experience smoother for slow devices were discussed (which most mobile devices fall into). With Web Workers it was found out how to offload computations to another thread so that they will not interrupt the main browser thread. This is very important when doing heavy client side applications like a mapping solution where path finding would be done on the client side.

Another technology discussed was the History interface that allows the web application to maintain the traditional user experience with working back and forward actions provided by the browser when doing single page applications. This can also be leveraged when doing multi-step forms if the framework of the server in the background does not handle them well. For example, this is a huge problem with stateless high performance web frameworks where developer would have to resort to storing the information from the previous steps in hidden fields in the form when moving from one step to the next one.

After these the more visible new CSS3 technologies that provide new and more efficient ways to handle layouts on multiple different screen sizes and forms while still using the same HTML content were discussed. From these the Flexbox layout gives the ability to have elements inside a container lay out in a predictable way into rows (or columns if the Flexbox direction is horizontal) even when the screen size changes. This technology gives good control over all the margins, directions etc. that control how the content should be laid out. Flexbox layout can also control the order of the elements inside the container without touching the content of the page itself.

The last technology was Grid layout which does not yet have a very good browser support; however, as a technology it looks very promising. It allows the content on the page to be almost totally detached from the styling which has been very difficult without this if the layout is to respond to screen size changes. While similar layout concepts are present in many popular CSS Grid frameworks they rely on modifying the content to contain the rows which makes it very hard to do big layout changes when devices size changes. With them when the screen gets small enough developer usually relies on making everything stacked on top of each other which might not always be the wanted result.

In overall all of these technologies are worth taking a good look into when building new web sites. They allow developers to build applications that work well and look correct on any device faster.

References

Alexis Deveria. A 2014, Can I use CSS Grid Layout?. Global user stats. Accessed on 9 May 2014. Retrieved from <http://caniuse.com/css-grid>

Alexis Deveria. A 2014, Can I use Flexible Box Layout Module?. Global user stats. Accessed on 9 May 2014. Retrieved from <http://caniuse.com/Flexbox>

Alexis Deveria. A 2014, Can I use Session history management?. Global user stats. Accessed on 9 May 2014. Retrieved from <http://caniuse.com/history>

Alexis Deveria. A 2014, Can I use Web Workers?. Global user stats. Accessed on 9 May 2014. Retrieved from <http://caniuse.com/webworkers>

W3C. P 1999. Hypertext Transfer Protocol – HTTP/1.1. 8.1.4 Practical Considerations. Accessed on 9 May 2014. Retrieved from <http://www.w3.org/Protocols/rfc2616/rfc2616-sec8.html>

W3C. P 2014. CSS Flexible Box Layout Module Level 1 W3C Working Draft. Accessed on 9 May 2014. Retrieved from <http://www.w3.org/TR/2014/WD-css-flexbox-1-20140325/>

W3C. P 2014. CSS Grid Layout Module Level 1 W3C Working Draft. Accessed on 9 May 2014. Retrieved from <http://www.w3.org/TR/2014/WD-css-grid-1-20140123/>

W3C. P 2014. HTML5 Candidate Recommendation. 5.5.2 The History interface. Accessed on 9 May 2014. Retrieved from <http://www.w3.org/TR/2014/CR-html5-20140429/browsers.html#the-history-interface>

W3C. P. 2012. Web Workers W3C Candidate Recommendation. Accessed on 9 May 2014. Retrieved from <http://www.w3.org/TR/2012/CR-workers-20120501/>