Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Technologies

Diploma Thesis

Building a CSS Framework

Bc. Jan Jeřábek

Supervisor: Ing. Petr Benda, Ph.D.

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Jan Jeřábek

Informatics

Thesis title

Building a CSS Framework

Objectives of thesis

The first objective of the Diploma thesis is to analyse advantages and disadvantages of CSS Framework implementation and to compare it with traditional development methods using HTML5, CSS3 and Javascript.

The second objective is to analyse the trending CSS Frameworks and grid systems. Design and build own CSS Framework based on their disadvantages using modern frontend design tools & methods, including CSS responsive pre-compiler.

Methodology

The methodology of this study is based on analysis and synthesis of technical information resources dealing with selected issues. In the practical part, advantages and disadvantages of CSS Framework implementation will be analyzed and results will be compared with traditional development methods. Acquired knowledge will be applied to the creation of own CSS framework based on trending CSS Frameworks analysis.

The proposed extent of the thesis

40 – 60 stran

Keywords

Frontend, Responsive Design, Webdesign, CSS3, HTML5, Bootstrap, Javascript, SASS, LESS, UI

Recommended information sources

CASTRO, E. – HYSLOP, B. *HTML5 a CSS3 : názorný průvodce tvorbou WWW stránek.* Brno: Computer Press, 2012. ISBN 978-80-251-3733-8.

ECCHER, C. *Profesionální webdesign : techniky a vzorová řešení pro XHTML a CSS.* Brno: Computer Press, 2010. ISBN 978-80-251-2677-6.

MEYER, E A. – MEYER, E A. *CSS : the definitive guide.* Beijing ; Sebastopol, CA: O'Reilly, 2007. ISBN 0596527330.

SCHAFER, S M. *HTML, XHTML a CSS : bible [pro tvorbu WWW stránek] : 4. vydání.* Praha: Grada, 2009. ISBN 978-80-247-2850-6.

TEAGUE, J C. *DHTML a CSS pro World Wide Web : praktická vizuální příručka.* Praha: Softpress, 2005. ISBN 80-86497-77-1.

Expected date of thesis defence

2016/17 SS – FEM

The Diploma Thesis Supervisor

Ing. Petr Benda, Ph.D.

Supervising department

Department of Information Technologies

Electronic approval: 18. 10. 2016

Ing. Jiří Vaněk, Ph.D.

Head of department

Electronic approval: 24. 10. 2016

Ing. Martin Pelikán, Ph.D.

Dean

Prague on 02. 01. 2017

Declaration

I declare that I have worked on my Master's thesis titled "Building a CSS Framework" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the diploma thesis, I declare that the thesis does not break copyrights of any third person.

In Prague on 22nd March 2017 .....

Acknowledgement

First and foremost, I want to thank my supervisor Ing. Petr Benda, Ph.D. for his constant support and guidance throughout the process of writing the diploma thesis.

A special thanks goes to all the members of ESN CULS Prague and unighters s.r.o., for giving me the opportunity to work on the project unibuddy and making this diploma thesis a contribution to a real project.

Tvorba CSS frameworku

Souhrn

V teoretické části této diplomové práce je analyzován responzivní webový design za použití CSS3.

Dále jsou analyzovány CSS kompilátory, jejich funkce a výhody oproti konvenčnímu CSS.

V praktické části je detailně popsaný návrh kompletního CSS frameworku, jeho porovnání s ostatními frameworky a celkové výhody tohoto řešení.

Klíčová slova: Frontend, Responzivní Design, Webdesign, CSS3, HTML5, Bootstrap, Javascript, SASS, LESS, UI

Building a CSS Framework

Summary

In the theoretical part of this diploma thesis, there is given an analysis of responsive web design using CSS3

Then the CSS precompilers are analysed as well, including their functionality and advantages over traditional CSS.

Practical part includes a design of a CSS framework, described in detail, its comparison to another frameworks and overall pros of this solution.

Keywords: Frontend, Responsive Design, Webdesign, CSS3, HTML5, Bootstrap, Javascript, SASS, LESS, UI

"The future is deterministic in principle, but not in practice."

— *David Gilmour*

Contents

Chapter 1

Introduction

Since the 1990, when the first html website was published in CERN by the British scientist Tim Berners-Lee[21], web design has made a long evolution journey. Internet websites have become a standard form of how people find information, communicate with each other and even how they make money.

Nowadays, web users are accessing websites through multiple devices which are mass-produced by technology leaders in the market. All the devices - computers, phones, tablets are programmed to read the website content and show the content on the display[4].

Since there are many methodologies of web-design and this area is evolving so fast, it is necessary to keep up with the newest trends. However, the basic languages used for web design are HTML, CSS and JavaScript, despite how often updated, compiled or generated. This thesis will introduce the newest possibilities how to handle these languages easier than it was by the time they were being invented.

Chapter 2

Objectives and Methodology

2.1 Objectives of the Thesis

The main objective is to develop a reusable front-end framework. The Framework will be created after an analysis of existing solutions that are available in early 2017. The results will guide the development, combining the strengths, whereas avoiding the weaknesses, which shall lead to a general improvement. The analysis will be made on grid systems, typography and overall experience.

The framework will consist of a responsive grid system, a set of reusable *mixins* and typography settings. The framework will focus on reusability in future projects and ability to be extended - to add more modules and elements in future.

The framework has to be customizable, in order to serve various purposes, from a simple static web presentation, to a complex information system. The core of the framework must be universal enough, customization must be easier than creating another framework/design for a different use-case. Nevertheless, the framework also has to serve the required purpose and not have any unnecessary/unused parts which would make the

size undesirably large.

The partial objectives are:

- to analyze methods of creating responsive layouts using CSS3 and CSS precompilers.
- to give a comparison between conventional CSS code and precompiled CSS code and/or existing CSS frameworks.
- to prepare the proposed CSS framework for implementation

2.2 Methodology

In theoretical part is an analysis of key parts of the modern web design.

Practical part is focused on building a new CSS Framework using the knowledge gained in the theoretical part. The framework follows the OOCSS (Object-oriented CSS) methodology.

2.2.1 OOCSS (Object-oriented CSS)

Object oriented CSS is a methodology of writing reusable CSS that is scalable and maintainable. OOCSS aims at making CSS more modular and scaleable. OOCSS was introduced by Nicole Sullivan at Web Directions North in 2009, the object oriented concept comes from more traditional engineering practices that are used in programming languages like PHP, Ruby or JavaScript. But objects in front-end development are simply HTML elements. The CSS is the place where those elements are modular to be able to be placed anywhere on a page and behave predictably. And this is done by following to Sullivan's two main principles: separating structure from skin and container from content.

Separating structure from skin means to abstract the structure and positioning styles of an object from the presentational styles, or skin.

Separating container from content means to break components' dependency of their containers. Any object should be able to be placed in another container and still look and behave the same.

Chapter 3

Theoretical Part

3.1 CSS3

CSS3 is the latest evolution of CSS language, which is recommended by World Wide Web Consortium (W3C)[1]. Its initial version was released in 1996.

3.1.1 Versions

CSS3 has been in development since the previous version release and the first drafts were published in 1999. Despite all the features that CSS1 and CSS2 brought to the web design, there were yet many features that were not possible to implement easily or misunderstood. Often using various hacks or workarounds which were not making these features impossible to implement, however it usually involved lots of nested divs, use of images instead of native UI elements or even technologies like Flash (eg. Rounded corners, color gradients, or animations). Not mentioning the mount of additional work required: a single color change required opening an additional graphics/animation software and saving the new set of assets

[1] http://www.w3c.org

Furthermore, there were couple of features which were not supported (or at least standardized) entirely. Probably the most used is the support of font embedding, which could have been done only using JavaScript

Another example is a multiple column layout. In CSS1/CSS2/CSS2.1 it could be done only using floated divs next to each other. But that is not the intention of floats, floats are designed to position an image within a text. CSS3 brings more methods to make a multi column layout - with consideration of height of the element (CSS3 Flex), or text wrapping to another column, making the newspaper-style article (CSS3 Multiple column)[11][5].

3.1.2 Development

Until the CSS version 3, the specification was developed as a whole bundle of new features and updates. The development of one version took couple years to be standardized[15]. This caused compatibility issues, because some browser developers pledged to (at least partly) support single CSS functions as they were being developed (Google, Mozilla, Opera, Apple), on the other hand, Microsoft Internet Explorer did not support new CSS features until the whole specification was released by W3C[13]. This caused issues to all developers, who wanted to work with new and easy-to-use features, while more than 70% of users were using CSS3 incompatible Internet Explorer in 2005,which is a significant number of potential website visitors and cannot be omitted[2].

Since the specification development has tended to be taking a significant time - CSS3 has been in development for more than 20 years now[11], W3C decided to split the development to separated modules, each of which is developed and released separately[15]. This allows to progressively implement finished modules in the browsers' compatibility[5].

3.1.3 Browser compatibility

Luckily for web developers, since January 2016, Internet Explorer have changed its update policies and instead of maintaining multiple versions, where each version has completely different compatibility with CSS specifications, their users are now forced to update to the newest and officially supported version[17].

This is a good step for both sides - developers, who are only building and maintaining (including security updates) just one version and on the other hand users - who have always the newest, therefore the most compatible version of their browser installed. This is a good reason for developers to use the newest CSS features. Other browsers have always force their users to update to the newest version[22].

Since Internet Explorer has always had problems with compatibility with CSS (which is nowadays almost non-existent in the newest IE versions / Edge), along with decreasing number of old version IE users, it is possible to use the full scope of CSS3 features[18][14].

3.2 Measurements units

CSS uses for its implementation various units for measuring length. Each unit has its best usage.

3.2.1 Absolute Lengths

The most used are **Pixels** (px). One pixel is exactly one dot on the screen. In modern high density displays, one pixel does not have to necessarily mean exactly one dot, but it is normalized to a chunk of dots, because one dot would be almost invisible to human eye.

Other absolute lengths are physical, but converted to px using the constant values. These are **Centimeters** (1cm = 37.8px), **Milimeters** (1mm = 3.78px) and **Inches** (1in = 96px)[16].

3.2.2 Font-Relative Lengths

To maintain responsiveness and accessibility allowing safely change the entire page's font size the unit **Em**, originally defined as a width of capital letter "M", presents the scale of current font-size. The default value of 1em is 16px. Em unit scales the current font size (0.5em = 8px) and its initial value is changed every time the font-size property is changed[16].

This behavior could cause confusion in nested elements with different font sizes. This can be solved by using the **Rem** units, instead of Em. Rem stands for "root em" and it always keeps the value of font-size of the root element of the page, instead of overwriting itself every time the font size is being changed. However, Rem units do not work in IE8, Safari 4 or IOS 3.2[16].

3.2.3 The Viewport Percentage Lengths

The Internet browser window, no matter if it is on computer, tablet or phone has its width and height of the visible area, which changes with window resize, or switching phone between portrait and landscape. This size of visible area is called "Viewport" and it can be described with percentage units **vw** (Viewport width) and **vh** (Viewport height), where 100 is full Viewport size[16]. Viewport lenghts are always absolute to Viewport size.

3.2.4 Percentage Length

Percentage (%) is technically not a unit, but a fraction of the full size. Despite of Viewport, which always shows an absolute size of the visible area, percentage value is a relative value to parent element size[16].

It is possible to find these similarities in Em and Rem.

3.2.5 Combination of Lengths in CSS3

Sometimes it is required to use absolute lengths along with percentages or font-relative lengths, often used in Hybrid Layouts.

For this reason, CSS3 allows to calculate with multiple lengths using the function calc(). Each of the length, running through the calculation is recalculated to corresponding Pixel value[9].

```
.box {
    width: calc(50% - 10px);
    }
```

An example above calculates half the size of the parent element and substracts 10px. Supported are operators +, -, * and /.

3.3 Default CSS3 rules

By default, every HTML element has its own default preset of CSS rules, without need of touching the style sheet.

Background would be white, text color would be black, h1 heading would have twice the size of the normal text and more others. These are basic values that are defined by W3C.

In some cases it is possible to take an advantage of this behavior and then it is not necessary to explicitly redefine the same rules and save some space of the code. However there are many browsers that would interpret these default settings differently.

In some cases it is required to redefine the whole default preset and start-over with the complete CSS definition.

That is where CSS Resets come in handy. They are short, sets of CSS rules that reset the styling of all HTML elements to a consistent baseline[10].

3.3.1 Eric Meyer's "Reset CSS" 2.0

One of the first CSS Reset tools was created by Eric Meyer, still being used by millions of websites. It comes with free licence and the author encourages developers not to use the tool as it comes, but to edit it co be in compliance with the user style sheet.

The CSS Reset clears all default rules, so they can be re-declared again which will cause great cross-browser compatibility[10].

Nothing more than a user defined style sheet will be displayed, which also brings noticeable disadvantages: each HTML element has to be re-declared and that comes with a lot of redundant code. What was once removed is added back again. This can be easily fixed by altering the CSS Reset code and omitting the parts that are used later in the code as the author suggests[10].

3.3.2 Normalize.css

Normalize.css is an alternative to the CSS reset created by Nicolas Gallagher and Jonathan Neal. It is considered as a more modern way of resetting CSS, currently used by Twitter, TweetDeck, GitHub, Soundcloud, Guardian and many more popular websited. The key difference to CSS reset is that instead of clearing all CSS rules completely, it sets all rules to match the W3C default rules.

Because the default rules are usually interpreted differently among the browsers and cause very basic incompatibilities, Normalize.css solves these problems and "normalizes" the default CSS to the same level for all browsers[6]. The source code of Normalize.css is in the Appendix B.

3.4 Responsive design

3.4.1 Layout

Layout is the way how the elements on page are arranged and what behavior they follow. There 4 basic types of layouts:

- Fixed-width Layout
- Fluid Layout
- Elastic Layout
- Hybrid Layout

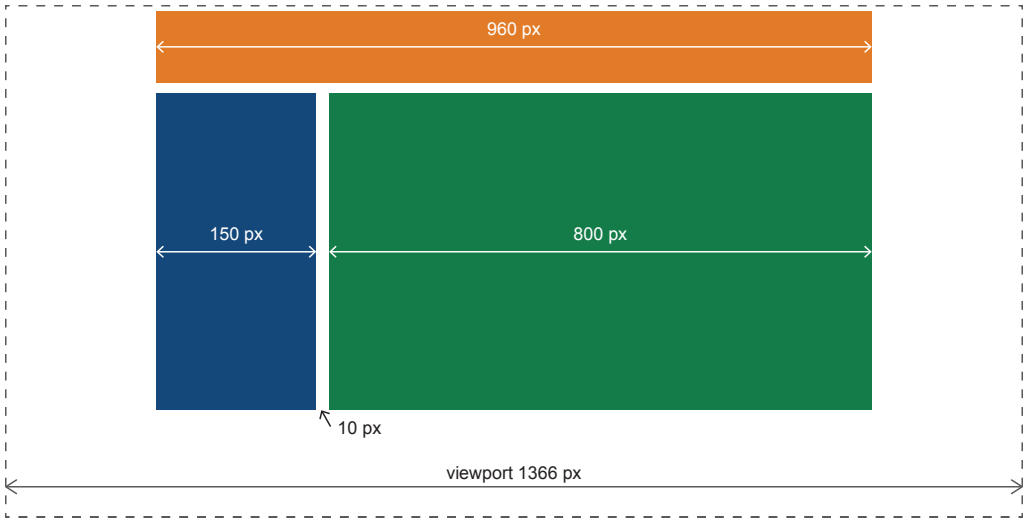
3.4.1.1 Fixed-width Layout

Layout with fixed width has its width limited with specific pixel value, as shown on the Figure 3.1. The most common value of 960px, as written by Cameron Moll in 2006, was considered as the most universal, because is divisible by numbers 3, 4, 5, 6, 8, 10, 12 and 15. However, the width of 960px was designed for the display resolution width of 1024px[9].

The advantage of this method is the pixel perfect precision. That can be useful if parts of the website are external modules, provided by the third party, with fixed size.

The method of fixed width does not take the current Viewport (mentioned in section 3.2.3) in consideration. This means, that significantly larger Viewport would show the layout as rather smaller portion of the screen, which does not effectively use the blank areas. If the Viewport would happen to be smaller than the fixed layout, the lack of space would be compensated with horizontal scrollbars, making the navigation on the page unpleasant[9].

Figure 3.1: **Fixed-width Layout composition**



Source: Own drawing

Nevertheless, this described method of a layout design is highly outdated, because while in 2006 the resolution of 1024px was used by almost 60% of Internet browsers, in 2016 it is only 3% of Internet browsers, according to W3C statistics. Nowadays, the most common resolution goes from 1366px up to 4K[12].

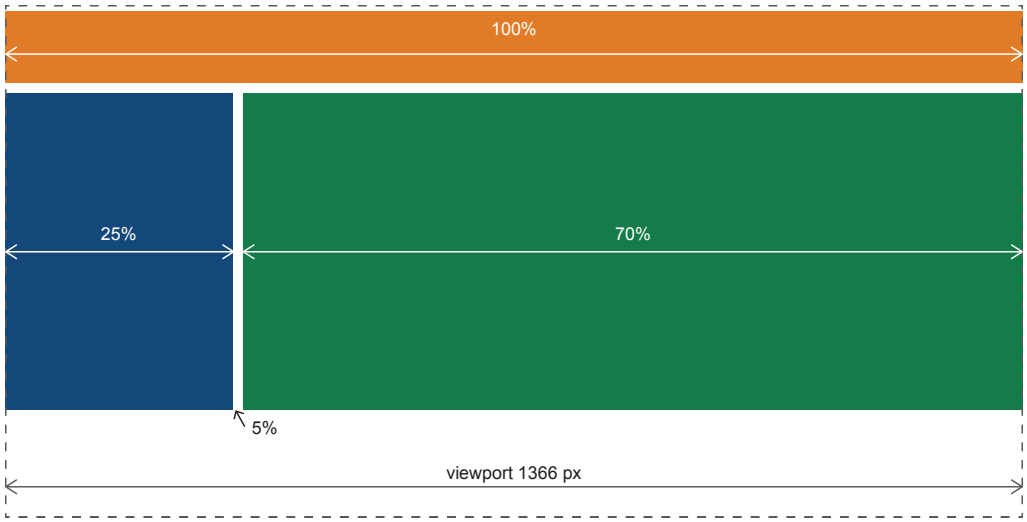
3.4.1.2 Fluid Layout

Fluid (often also called "Liquid") layouts use percentage values instead of absolute pixel values. The layout scales according to the actual Viewport. It does not matter if the Viewport is 2560px wide laptop screen, or 768px wide tablet screen[9].

Single parts of the layout are sized using the Percentage lengths or Viewport percentage lengths (mentioned in section 3.2). As long as the sum of all layout components does not exceed 100vw or %, there would not be any scrollbars (as long as the content does not overlay)[2].

This method deals with the Fixed-width Layout problem, however,

Figure 3.2: **Fluid Layout composition**



Source: Own drawing

using this method without Media queries (explained in section), or without aware of the font size would still lead to unwanted results. The larger displays would stretch the page to be very wide, causing the text paragraphs extremely wide and very hard to read. The mobile browsers would, on the other hand, shrink the layout to very narrow and unreadable[9].

3.4.1.3 Elastic Layout

The problems with Fluid Layout, causing very wide sections of text can be solved implementing an Elastic Layout. Instead of Viewport or Percentage units, the sizes are specified with em or rem units. This assures that the Layout would keep proportions relative to the font size[9].

According to Robert Bringhurst, the best readability of any document is assured when the paragraph width is between 45 and 70 characters[1]. Therefore, making the content area around 55em wide would improve the overall user's reading experience.

Elastic Layout also scales the layout with user-defined font size, keeping the characters per line number. This improves overall accessibility to people with sight disorders[9].

Sadly enough, this solution does not solve the issues caused by the Fixed-width Layout, moreover it extends these issues, because the behavior of the layout could be unpredictable[9].

3.4.1.4 Hybrid Layout

So far every solution has come with their specific advantages and disadvantages. It is hard to state which solution is the best or worst, because their strengths and weaknesses do not usually correspond with each other.

The solution of this dilemma is to use the advantages of each solution, retain the accessibility principles and mainly, use the Media Queries (explained in the section 3.4.2), to minimize the negative effects of all layouts

Hybrid Layout uses Viewport percentage where it is needed to allocate certain portion of the Viewport, Em sizes for the parts which should scale with font-size and pixel sizes for elements which require constant size.

3.4.2 Media Queries

Media queries allows to make the browser evaluate an expression. If the expression is true, the browser will process the CSS code enclosed, if not, it will left ignored.

Media queries are constructed of media type, expression and a piece of CSS code, to be processed in case the expression is true. Media types are shown in the table 4.1

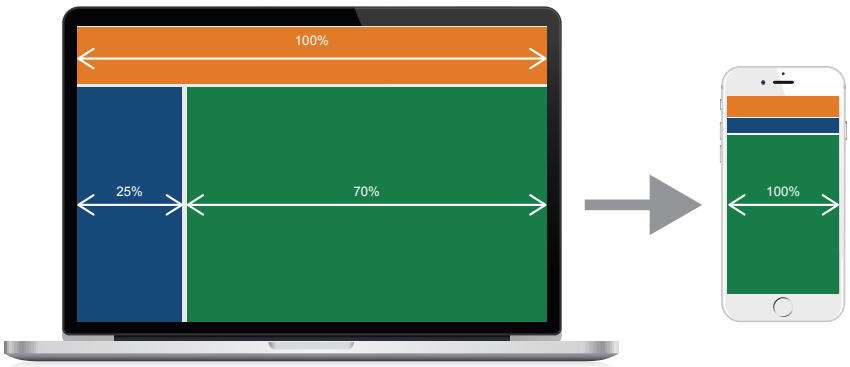
Table 3.1: **Media types and their scope**.

Type	Target devices
all	All devices (default)
aural	Used for speech and sound synthesizers
braille	Used for braille tactile feedback devices
embossed	Used for paged braille printers
handheld	Used for small or handheld devices
print	Used for printers
projection	Used for projected presentations, like slides
screen	Used for computer screens
tty	Used for media using a fixed-pitch character grid, like teletypes and terminals
tv	Used for television-type devices

Source: http://css-tricks.com/snippets/css/all-stylesheet-media-types

Most used media types are all, screen and print. Event most of the handheld devices support only screen type, because developers do not usually distinguish between media types[9].

Figure 3.3: **Using media queries**



Source: Own drawing

The basic approach to page responsiveness is to test Viewport for width. To actually test the value, width is prefixed with min- or max-. There can be used even multiple expressions, separated by logical operators like and, or, not, only.

```
@media screen and (min-width: 500px) {
    /*
    Any CSS rules here will be processed only, when
    the Viewport width is more than 500px.
    */
}
```

3.4.3 Desktop first approach

Desktop first approach is the traditional way design that has been practised since the beginning, because the first devices using Internet browsers were computers.

Global CSS rules are assigned to work for desktop (larger screen) and as the screen size decreases down to mobile size, additional CSS rules are applied to overwrite the global ones[9].

```
.box {
    width: 25%;
    float: left;
}

@media screen and (max-width: 768px) {
    .box {
        width: 50%;
    }
}

@media screen and (max-width: 480px) {
    .box {
        width: 100%;
        float: none;
```

```
    }
}
```

In example above, the element with class "box" would take 25% of the parent element, which is 1/4 of the space. In other words, it is possible to place 4 boxes next to each other. If the Viewport size is lower than 768px, the width is overwritten with 50% (only 2 boxes next to each other). If the Viewport is even smaller, less than 480px, the both width rules are overwritten with 100%. This brings scalability from large desktop, to tablet and phone.

3.4.4 Mobile first approach

Mobile first approach puts the basic set of rules for small mobile version and media queries are used to scale the design upwards. This also brings a great experience to users with incompatible browsers, readers and other devices for people with disabilities, because the basic design is simpler (because it is mainly for mobile) and easier to read.

According to the W3C Statistics of Operation Systems used for Internet browsing, the mobile devices have been becoming stronger and stronger way of consuming the Internet. Since 2011, their market share has been growing from 0 to current 6.3% (as of January 2017)[19]. This phenomena brings mobile devices to question.

Instead of the Desktop first approach where the full feature desktop version is scaled down and simplified, the mobile first starts with full functional mobile version, enriching desktop versions with more functionality.

Both principles may sound the same with a different point of view, however, the Mobile first approach focuses on the best experience on mobile devices.

```
.box {
        width: 100%;
        float: none;
    }

@media screen and (min-width: 768px) {
    .box {
        width: 50%;
    }
}

@media screen and (min-width: 1200px) {
    .box {
        width: 25%;
        float: left;
    }
}
```

At this example, the box would maintain the same behavior as in the example in the Section 3.4.3. The main difference is scaling the boxes from the single 100% width box, up to 4 smaller 25% width boxes. This is achieved using min-width instead of max-width in media queries and with reversed order of rules (Rules are executed from the top to the bottom).

3.4.5 Breakpoints

When designing a responsive website using media queries, no matter whether using Mobile or Desktop first, it is needed to separate the design to look different on different devices - mobile phone, tablet (landscape/portrait), computer (small screens/large screens).

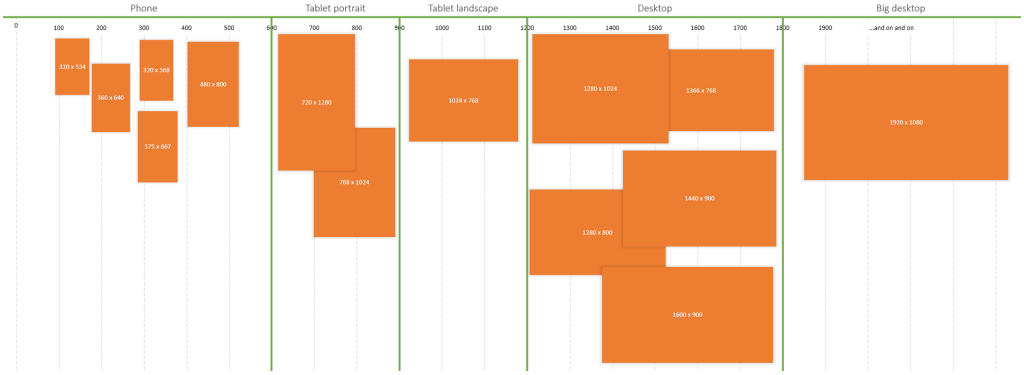
As mentioned before, in the Section 3.4.2, the separation uses specific the Viewport width, which is evaluated with operators min-width and max-width. The widths, which break the layout are called Breakpoints[9].

Breakpoints have to be chosen carefully, because small number of

breakpoints would cause insufficient breaking, leading to bad experience with multiple devices. Mobile version could display on tablet and one version for computer screen would display for 11" and 30" displays. High number of breakpoints would fit perfectly every device and resolution, but it would cause a lot of redundant code and every rule would be defined many times.

To choose balanced number of breakpoints, David Gilbertson wrote an article from November 2016, where he arguments his approach of making breakpoints.

Figure 3.4: **Breakpoints by David Gilbertson**



Source: https://medium.freecodecamp.com/the-100-correct-way-to-do-css-breakpoints-88d6a5ba1862

According to him, the breakpoints should not be defined as single points that break the design. He chose a different approach - using the StatCounter global stats of resolution usage he identified groups with the most common resolution with accordance to their usage.

It may not seem as a big difference to the conventional naming, where one breakpoint is called "phone" or "small", then up to the "tablet" or "medium" and "computer" or "large". The breakpoints - single points themselves do not define the screen size. It is the range, within the group of devices with common screen size. In other words there is not one breakpoint

at 600px, called "phone", but there is a range from 0 to 600px, called "phone".

He also recommends to be declarative and not to name the groups as "small", "medium", "large", because it does not make any sense. He encourages to name the groups as "phone", "tablet", etc.

Ultimately, using the StatCounter data, he defined 5 groups that satisfy the most of the devices, while the number of 5 is still convenient enough for development[7].

The groups are defined as following:

- **Phone** 0px - 599px
- **Tablet portrait** 600px - 899px
- **Tablet landscape** 900px - 1199px
- **Desktop** 1200px - 1799px
- **Desktop large** 1800px and more

3.5 CSS Precompilers

CSS is in fact very primitive language. The very basic idea of CSS is to select an element from the DOM using wide range of selectors and pseudo-selectors and simply apply a set of styling rules. While rendering, the browser looks for matching DOM elements and selectors, displaying them as their CSS rules are set.

CSS offers pseudo-selectors that are able to process primitive events, like :hover (matches rules on mouse over), :checked (when a checkbox is checked), form validation rules or operators AND, OR, NOT.

However, CSS does not support features that are commonly available in higher-level programming languages, or that are simply missing in plain CSS. These features are available in CSS Preprocessors:

- Variables
- Nesting
- Mixins (custom methods)
- Extends
- Color Operations
- If/Else Statements
- Loops
- Mathematic Operations
- Imports

Since there are these features not available in CSS, many parts of code becomes redundant because everything has to be explicitly stated. This is opposing to the DRY (Don't repeat yourself) idea, which every programmer should follow[8].

There are 2 major CSS precompilers used in 2017 - SASS and LESS.

3.5.1 SASS vs. LESS

SASS was introduced in 2006 as Style sheet language. Later, its creators Hampton Catlin and Natalie Weizenbaum continued with development and made SASS a scripting language implemented in Ruby.

The first syntax (*.sass files) was based on HAML (or YAML), which is relies on new-lines and indenting.

Figure 3.5: **SASS Logo**

Source: Wikimedia Commons

```
$color: #222222

.text
    color: $color
    font-weight: bold
```

The original syntax omits semicolons and parentheses. However, developers were mostly struggling with this syntax, because everyone was used to CSS syntax. That is why the creators decided to take an inspiration from LESS and created a new css-like syntax called SCSS (*.scss files)[8].

```
$color: #222222;

.text {
    color: $color;
    font-weight: bold;
    }
```

Nowadays, the SCSS syntax is used by majority of developers[3].

LESS was was created as a response to confusing syntax of Sass by Alexis Sellier. It is highly inspired by it, but it had brought a very clean

syntax, where LESS functions are included in plain CSS syntax. LESS is implemented in JavaScript[3].

The syntax is very similar to SCSS, because SCSS was actually influenced by LESS

```
@color: #222222;

.text {
    color: @color;
    font-weight: bold;
    }
```

At this example there is obvious similarity of the syntax (variables are prefixed with @ in LESS, with $ in SASS). Though, there are more syntactic differences in If/Else Statements or Loops.

Figure 3.6: **Sass Logo**

Source: Wikimedia Commons

According to Chris Coyier from CSS-TRICKS.com, the functionality of SASS and LESS are very comparable - both precompilers can handle all the features mentioned in the Section 3.5.

SASS have implemented better methodologies of extending classes, that are more considerate to the output CSS code, which is eventually cleaner than in LESS. Also the previously mentioned dollar sign, used as a prefix in SASS has no meaning in plain CSS, while the @ symbol, used by

LESS is also used by plain CSS for declaring media queries or keyframes in animations.

Coyier also claims that SASS has had larger community of developers, since both projects are open-source hosted on Github. Comparing Pull Requests, Commits, Issues discussion and Releases[3]. For this reason, following Sections are focused on SASS and all examples of the basic functions are in SCSS syntax.

3.5.2 Variables

Layouts are usually build using precise metrics (lengths and colors) that are repeated within the code multiple times as specified, or derived from the original value.

Many layouts use a narrow set of colors that are repeated many times in the code. This does not comply with the DRY principles, because when the website owner decides to change one color to a different one, it is necessary to change all references of this particular color in the code (Using the text editor's Find/Replace functions). In long style sheets, there might be hundreds of references to one particular color.

This issue is solved by implementing variables. As in higher-level programming variables, in SASS, variables are used to store a value, which can be printed on multiple places in code. Considering the previous example, to change one color in whole style sheet is a matter of changing the color value in the variable declaration.

SASS also supports saving values in simple arrays. These arrays can be used for iterating using loops[8].

```
$color: #FFCC55;
.button {
    color: $color;
    /* Color will be #FFCC55 as defined above */
    border: 1px solid $color;
    }
```

3.5.3 Nesting

Since HTML has a hierarchy of elements - DOM, it is necessary to consider this hierarchy in CSS as well. In CSS this is possible to be done by using multiple selectors:

```
.container { width: 50%; }
.container ul { list-style-type: none; }
.container ul > li { display: inline-block; }
.container ul > li:hover { background: black; }
```

This, however, brings a lot of writing, copying and most importantly a lot of code repetitions. In SASS, selector hierarchy can be preserved by nesting selectors in their parents. Nesting is also possible for whole media queries for responsive design.

```
.container {
    width: 50%;
    ul {
        list-style-type: none;
        > li {
            display:inline-block;
            &:hover {
                background: black;
            }
        }
    }
}
```

This would compile to the same CSS output as in the previous example. As may be seen in the example, the :hover pseudoselector is prefixed with & sign. This is for including the parent's hierarchy into the selector[8].

3.5.4 Extending

Extending keeps SASS code semantic. When it is needed to have a certain element duplicated from existing one and even altered, it is possible using @extend. Extended element will clone all rule to another selector, while maintaining output CSS as simple as possible. This, however is not possible when both selectors lie in different media scopes[8].

```
.button-default {
    background: #FFF;
    border: 1px #CCC;
    color: #222;
    text-align: center;
}

.button-blue {
    @extend .button-default;
    background: #FF8;
}
```

.button-blue will clone all properties from .button-default and overwrite the background color with #FF8.

Extending makes code more clean and easier to edit[8].

3.5.5 Mixins (custom methods)

Mixins allows to take a fragment of SASS code and include everywhere in the code - just like @extend. But Mixins are more powerful, because instead of extending a class' properties, a Mixin can be any fragment of code. This is powerful for properties that are repeated multiple times among the

code. A Mixin can also accept variables as arguments, which make them scalable for many purposes. Especially for maintaining browser compatibility, because certain CSS properties have different expressions for a different browser[8].

```
@mixin rotate($deg){
  -webkit-transform: rotate($deg);
  -moz-transform: rotate($deg);
  -o-transform: rotate($deg);
  transform: rotate($deg);
}

.rotate-right {
    @include rotate(90deg);
}

.upside-down {
    @include rotate(180deg);
}
```

Mixins also allows to create multiple color variations of same design - the only parameter for certain fragment could be the color and the rest of the code is generated by SASS

3.5.6 Color Operations

SASS is capable to work with colors or color-based variables and process various color functions[20]:

- **mix** Mixes two colors together
- **lighten** Lightens up by given percentage
- **darken** Darkens up by given percentage
- **saturate** Saturates color by given percentage

- **desaturate** Desaturates color by given percentage
- **rgba** Adds alpha (opacity) to a color

With these functions it is not necessary to calculate colors manually when it is needed to use eg. lighter color on hover. For SASS If/Else and loops is also possible to get lightness, saturation, hue, alpha value of given color[8].

3.5.7 If/Else Statements

Control expression can help making decisions according to matched conditions. Supported are If, Else, and Else if. It is possible to use mathematical conditions or functions that return a value to match the statement[8].

```
@if lightness($color) > 30% {
    background-color: black;
}

@else {
    background-color: white;
}
```

3.5.8 Loops

SASS supports loops as they are known in higher programming languages.

Loops are extremely useful for using arrays of colors. Allows to iterate the array and apply a color to a selector. When combined with nth pseudo selectors or interpolation, iterating through an array can save a lot of manual writing of code. Also, many cases would be impossible to implement in CSS without loops[8].

EACH loop iterates every element of an array while executing the code inside of the loop. It takes the form @each $var in list.

FOR loop considers the index in every iteration. This is useful when working with nth pseudo selectors. The syntax is @for $i from 1 through 4.

WHILE loop iterates as long as the condition has not been fulfilled, similarly to if. The condition is written as @while $types == 0.

3.5.9 Interpolating

Interpolation allows to construct selectors by placing variables in their names or pseudo selectors. Often, they are combined with loops, since it is possible to create new selectors each iteration.

To print any value anywhere in the code, it is necessary to write the variable wrapped in braces, prefixed with pound sign - #{ $variable }[8].

```
$car: skoda;

.car_#{$car} {
    display: block;
}
```

3.5.10 Mathematic Operations

SASS can handle standard arithmetic operations when working with numbers. It supports unit conversion. There are also functions for working with numbers[8][20]:

- **percentage** Converts a unitless number to a percentage.
- **round** Rounds a number to the nearest whole number.
- **ceil** Rounds a number up to the next whole number.

- **floor** Rounds a number down to the previous whole number.
- **abs** Returns the absolute value of a number.
- **min** Finds the minimum of several numbers.
- **max** Finds the maximum of several numbers.
- **random** Returns a random number.

3.5.11 Imports

Rather than using a one large file, separating code in multiple small pieces is helpful for expressing the declarations and increasing maintainability and control over the code. For the code maintenance it is better to keep all numeric or color values in variables and all variables having stored in separate file. Code should be split in files according to the purpose of the code in order to keep code semantic[8].

To import a file, the @import expression is used. SASS does not require to specify the file extension, it automatically searches for *.scss and *.sass extensions.

Chapter 4

Practical Part

4.1 Existing Solutions

Nowadays, there are many frameworks, that are available. They all have their own place on the market, but all of them balance on size, universality, functionality and compatibility.

The objective of this work is to analyze their strengths and weaknesses and use these information to build a new framework that will minimize weaknesses of 3 major CSS Frameworks.

The Major frameworks are

- Bootstrap[2] by Twitter
- PureCSS[3] by Yahoo
- Foundation[4] by ZURB

[2]http://getbootstrap.com
[3]https://purecss.io
[4]http://foundation.zurb.com

4.1.1 Bootstrap

Bootstrap uses Floating grid, which can be in 2017 considered as outdated, although, Bootstrap aims on compatibility.

Bootstrap also does not support font-aware units - ems and rems. All lengths are made using pixels, which makes whole design lot less responsive and accessible. In future version 4, the Flexbox will be added and part of compatibility omitted.

On the other hand, Bootstrap benefits from massive community and high quality standards - new version has been in development for more than a year and by the beginning of 2017 it is still in its Alpha version.

Bootstrap aims at less skilled developer group - with small effort it brings a professional look of the layout, while keeping great compatibility and gentle learning curve.

The main disadvantage of Bootstrap is its homogeneous design, despite of its modern design, the design keeps repeating and experienced designer can recognize Bootstrap design (which is not necessary a bad thing). These designs lack creativity.

The proposed CSS Framework should not be considered as a replacement of CSS and Web design work, but it should help developers to create websites more easily and more consistent.

Bootstrap 3 is, again, very outdated for the absolute lengths and floating system. Its Github repository shows that the last update has been made in July 2016. Bootstrap version 4 has been in Alpha version since 2015 and it cannot be expected to be finished any time soon. Version 4 also keeps backward compatibility with version 3, which makes the final CSS file even larger.

4.1.2 PureCSS

PureCSS, is on the other hand a very light-weight framework, which makes it a great for comparison with proposed CSS Framework.

Its grid system is made using floating system, instead of CSS3 Flexbox, which should not be ignored.

Overall, the aim of PureCSS is in compliance with proposed CSS Framework, since it aims at experienced developers. Other than the grid system is only serves with basic typography and form components.

Its official Gihub repository has not been updated since November 2016 and there is no known developing progress on it currently. For this reason it could be considered as abandoned project.

4.1.3 Foundation

Foundation is a very complex front-end framework that is as fundamental as Bootstrap. It aims to more experienced developers.

The grid system is sill floating by default, however it allows to use Flex-based instead.

The typography is not responsive, which is the main weakness of Foundation and it would require further implementation.

Foundation offers many components that makes it a very universal (thus slightly robust) framework.

4.1.4 Comparison of the CSS Frameworks

The following table shows the comparison of 3 major CSS Framework, summarized in the table:

Table 4.1: **Comparison of 3 major CSS Frameworks**.

	Bootstrap 3	**Pure CSS**	**Foundation**
Responsive Grid			
Technology	Float	Float	Float
Columns	12	24	(12)
Breakpoints	4	4	3
Responsive gutters	no	no	yes
Typography			
Responsive	no	partial	no
Typography units	px	em	em
Reset	reboot.css	Normalize.css	Normalize.css
Miscellaneous			
Responsiveness	Mobile first	Mobile first	Mobile first
CSS Precompiler	LESS	-	SASS
Size	143Kb	24Kb	100Kb
Difficulty	Easy	Medium	Easy
GitHUB stars	108k	16k	25k

Source: Own research

After a deep analysis of existing CSS frameworks, there are several good practises that should be included in the framework:

- Flexbox based grid
- Responsive Typography
- Responsive Gutters
- Variable number of columns
- Normalize.css
- Relative units

- Mobile First
- SASS precompiler

There are also bad practises or out-to-date features that should be ommited:

- Float based grid
- Pixel units
- Fixed number of columns
- Relative units
- Very old browser support
- No CSS precompiler

4.2 Framework Specification

In web development, there are various elements that have to be taken care of in every project. These elements are always the same, or more less similar. Copying, pasting and repeating the code are the main factors that are eliminated in CSS Framework, making it the major benefit of it.

Considering this, there are three major functions that a CSS Framework need to serve.

4.3 Structure

First of all is a Responsive Grid system. No matter how the design look like - no matter what theme the design uses, there is a core functionality, which manages the responsiveness and positions the parts of the layout.

The grid system will be Flexbox based. Thus, to maintain compatibility with old browsers and to follow modern web design trends, the framework will follow the mobile first approach.

The second part is a set of mixins that help to keep the amount of code at minimum using pre-coded parts or tools to include them to the project. These tools help to maintain code semantics while creating a unique design.

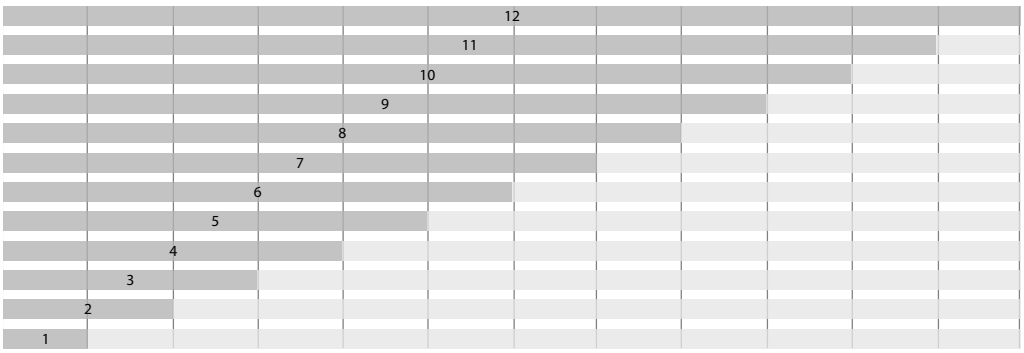
The third part is a set of typography settings. Typography has to be easy to manage and its changes has to be set accordingly within the project. This part is also connected to the first part, because typography is a key part of page responsiveness.

4.4 Grid System

Grid system is a core function of the CSS Framework. It manages positioning and responsiveness of all chunks of the layout.

The basic grid division that many CSS Frameworks follow is 12 columns. 100% of the container width is divided to 12 equal parts, defined by percentual portion of chosen number of columns. In other words, the chosen number of columns is a fragment of the total number of columns.

Figure 4.1: **12 column grid**



Source: Own drawing

4.4.1 Gutters

Gutters create the spacing between the individual items. In most designs they are important to make the content readable.

Gutters are defined by two halves: Container part and Item part. If gutters were defined only on items, the first and last column of items would have smaller gutters on the edge of the viewport.

```
/* grid-gutters by Jan Jerabek 2017 */


@mixin gutters($gutter){
  padding-top: ceil($gutter);
  padding-right: ceil($gutter);
  padding-bottom: floor($gutter);
  padding-right: floor($gutter);

  $mobile: nth(map-values($screen-breakpoints), 1);
  @media screen and (min-width: $mobile) {
    // mobile-up viewports have doubled gutters
    padding-top: ceil($gutter*2);
    padding-right: ceil($gutter*2);
    padding-bottom: floor($gutter*2);
    padding-left: floor($gutter*2);
  }
}
```

Each pair of gutters on the opposite sides have one gutter that is floored (Rounded down) and one that is ceiled (Rounded up). This prevents lose of space when the indivisible value of the gutter is provided, since pixels can not be decimal numbers (eg. gutter 2.5 is divided to 2 and 3, matching the sum of 2 x 2.5)

Gutter size is accepted for mobile Viewport as it comes. For larger Viewports the value is doubled. This has to be taken in consideration when choosing a gutter value.

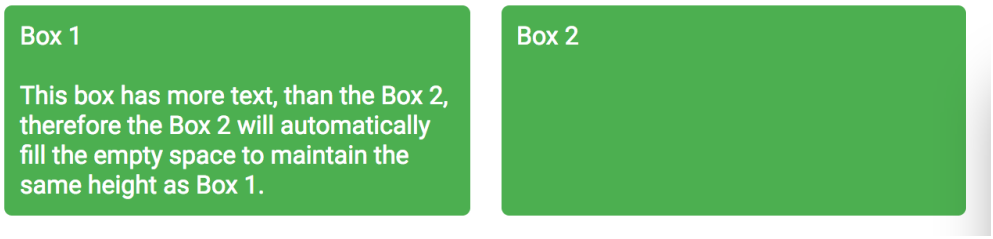
4.4.2 Container

Every set of items in Responsive Grid has to be wrapped in container . The container sets all the properties necessary for making the flexbox grid work.

The flexbox is configured to wrap the items if necessary. This enables to create multi-row containers, that will simply wrap overlapping items to a new row.

All items are aligned to top-left corner and items are not stretched to the free space - items maintain their width.

Figure 4.2: **Equal height of Flexbox items**



Source: Screenshot

Container also solves the problem with different height of items that would cause design inconsistency and most likely would have to be solved individually (eg. using JavaScript). Flexbox is able to stretch the child items to fit the largest item and therefore make all items equal height, as shown on the figure 4.2.

The code of the container mixin is following:

```
/* grid-container by Jan Jerabek 2017 */

@mixin grid-container($gutter: $grid-gutter) {
  @include display-flex;
  flex-wrap: wrap;
  -ms-flex-wrap: wrap;
  justify-content: flex-start;
  align-content: stretch;
  width: 100%;
  height: 100%;

  @include gutters($gutter);

  > div {
    // Making the child item another flex-container
    // helps with the compatibility with webkit browsers
    @include display-flex;
  }
}
```

4.4.3 Item

An Item is an element of the grid system. It contains a responsive container for another content - text, div, or even another nested grid.

As mentioned before, columns are defined as a portion of the total number of columns which is set as a variable attribute. The size of an item is calculated, as mentioned in Floating layouts (Section 3.4.1.2), in percentage lengths.

The width value is percentage calculated by as a fragment of desired size and total number of columns, converted to percents:

```
width: $size / $grid * 100%;
```

This calculation achieves that item with size of 6 out of 12 columns will have 50% width. 3 out of 12 will have 25%, etc.

This value is set explicitly as width, not as flex-basis value, which it technically should be as flex-basis, but to achieve best compatibility (especially with Internet Explorer 11).

The mixin for specifying an item looks as following:

```
/* grid-item by Jan Jerabek 2017 */

@mixin grid-item($size, $grid, $gutter) {

  padding-top: ceil($gutter * 2);
  padding-right: ceil($gutter * 2);
  padding-bottom: floor($gutter * 2);
  padding-left: floor($gutter * 2);

  // width has to be specified, because of IE11
  width: $size / $grid * 100%;

  // flex shrink and stretch are 0
  // flex basis calculates percentage of the column width
  @include flex(0, 0, auto);

}
```

The class which this mixin is applied to should have either constant size (including multiple breakpoints), or the classes generated for all breakpoints and sizes - the Class generator.

4.4.4 Class Generator

The class generator is present for developers who choose not to develop their own style sheet. For development it is easier - every class has self-explanatory name which includes a size of the element and which break-

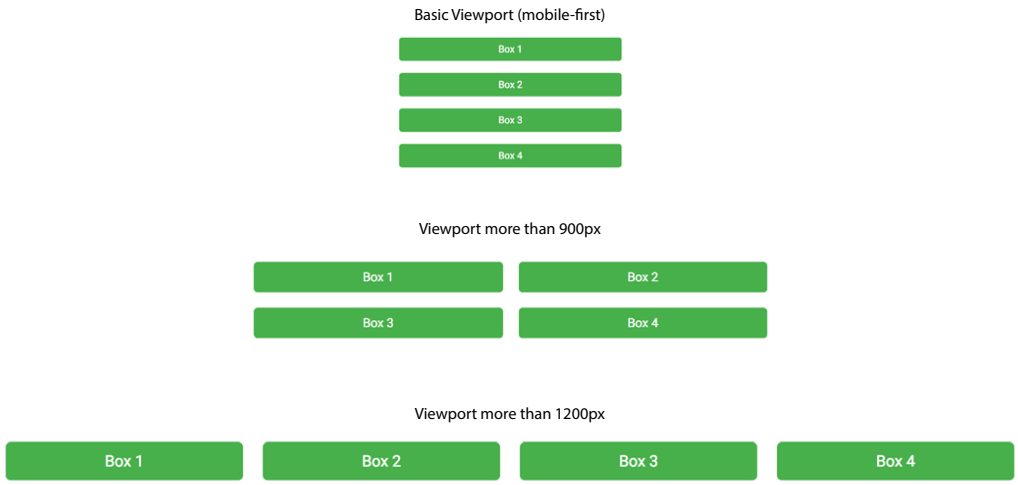
point category is the size for. The naming pattern is: item-%Breakpoint%-%Number%

HTML:

```
<div class="container">
    <div class="item-md-up-3"><div>Box 1</div></div>
    <div class="item-md-up-3"><div>Box 2</div></div>
    <div class="item-md-up-3"><div>Box 3</div></div>
    <div class="item-md-up-3"><div>Box 4</div></div>
</div>
```

As shown on figure 4.5, there are 4 boxes, that represent the grid system behavior. By default, they have full 100% width, in four rows. If the Viewport width is wider than 900px (md-up), boxes will be shown with 50% width, in two rows. If the Viewport is wider than 1200px (lg-up), boxes will be shown with 25% width, in single row.

Figure 4.3: **Generated classes**



Source: Screenshot

The generator is built using two loops. The first runs over all screen breakpoints defined below:

```
$screen-breakpoints: (
        sm-up: 600px,
        md-up: 900px,
        lg-up: 1200px,
        hd-up: 1800px
        );
```

The second loop runs n-times, where n is a total number of columns (12 by default).

For each combination of breakpoint and column number is @item mixin called and created appropriate class named using the naming pattern mentioned previously.

```
/* class-generator by Jan Jerabek 2017 */

@mixin class-generator($grid, $gutter) {

  [class*="item-"]{
    @include grid-item($grid, $grid, $gutter);
    // mobile-first width and gutter
    // default settings 100% width
  }

  // loop each screen breakpoint
  @each $screen in map-keys($screen-breakpoints) {

    // loop all grid column numbers
    @for $i from 1 through $grid {

      .item-#{$screen}-#{$i} {
        // then generate classes for
        // every other breakpoint and column size
        $media: map-get($screen-breakpoints, $screen);
        @media screen and (min-width: $media) {
            @include grid-item($i, $grid, $gutter);
        }
```

```
        > div {
          // IE fix
          width: 100%;

          @include gutters($gutter);

          //@include flex(0, 1, auto);
          flex: 1;
          @include display-flex;

          // Safari does not stretch flex child height.
          // Child is then turned to another flex
          // container to fix this issue
        }
      }
    }
  }
}
```

This generator will create 48 classes using flexbox grid system in total, each of which is responsive.

If class specified in HTML element is for wider breakpoint than is the current Viewport, ie. the class is specified only for desktop, the element will have 100% width on mobiles and tablets.

The grid system accepts the least defined breakpoint up to the higher viewport (if exists). All in compliance with mobile-first approach.

4.5 Mixins

The default library of mixins will enhance the development as it includes the most common tools for web design.

Some of the mixins are only providing backward compatibility with

older Internet browsers by adding so-called Vendor prefixes. These prefixes are for certain browsers only, from the time, when the functionality was not completely specified. Vendor prefixes can be also when compiling the code, using Autoprefixer tool. It can be added to Webpack. However, for the framework to be able to deployed in various ways, these vendor mixins are provided.

4.5.1 Clearfix

Since floating was not created to the purpose of floating , as mentioned in the Section 3.1.1, floated cause its container to collapse. Typical example is horizontal <ul> navigation, where the child <li> elements are floated. Therefore, the container has to be cleared by including to the parent <ul>. The clearing method was inspired by Nicolas Gallagher's Micro clearfix hack, but I decided to omit support for Internet Explorer 6 and 7.

```
/* Clearfix mixin by Jan Jerabek 2017
inspired by: http://nicolasgallagher.com/micro-clearfix-hack/ */
@mixin clearfix {
  &:before,
  &:after {
    content: " ";
    display: table;
  }
  &:after {
    clear: both;
  }
}
```

Using the :before and :after pseudo selectors prevents the need of a dedicated clearing element, which saves the amount of code.

4.5.2 Responsive Embed

When embedding a video, either from popular video hosting websites (such as Youtube). The embed code usually comes as an iframe, which is in its basis not a responsive object - it does not scale down with with the Viewport size. This mixin, included to a will calculate its height using given aspect ratio (if not explicitly given, the most common 16:9 will be used as default). The iframe will be absolutely positioned over this blank space. The padding-top property is used for older IE compatibility.

```
/* Embed mixin by Jan Jerabek 2017 */
@mixin embed($width: 16, $height: 9) {
  position: relative;
  /* Calculate padding by given ratio */
  padding-bottom: $height / $width * 100%;
  padding-top: 25px;
  height: 0;
  overflow: hidden;

  iframe {
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;
  }
}
```

4.5.3 Box Shadow

Box shadow, although it is single CSS property, it has multiple declarations with different vendor prefixes to maintain support of older browsers, where the box-shadow property was experimental.

The box-shadow mixin accepts arguments in same form as the box-shadow property does - vertical position, horizontal position, blur amount,

spread of shadow, color of shadow and an inset boolean. Including a single mixin will generate all vendor variants of box-shadow property.

```
/* box-shadow mixin by Jan Jerabek 2017 */
@mixin box-shadow($top, $left, $blur, $spread, $color, $inset) {
  @if $inset {
    -webkit-box-shadow: inset $top $left $blur $spread $color;
    -moz-box-shadow: inset $top $left $blur $spread $color;
    box-shadow: inset $top $left $blur $spread $color;
  } @else {
    -webkit-box-shadow: $top $left $blur $spread $color;
    -moz-box-shadow: $top $left $blur $spread $color;
    box-shadow: $top $left $blur $spread $color;
  }
}
```

4.5.4 Rotate

Rotate is a simple mixin, for generating vendor prefixes for CSS3 property rotate. The only arguments is degree in degree units.

```
/* rotate mixin by Jan Jerabek 2017 */
@mixin rotate($deg){
  -webkit-transform: rotate($deg);
  -moz-transform: rotate($deg);
  -o-transform: rotate($deg);
  transform: rotate($deg);
}
```

4.5.5 Linear Gradient

Linear gradient mixin, with vendor prefixes, supports all major browsers. The arguments are used in the same way as the linear-gradient CSS property. There are two colors to be set as well as angle of gradient in degrees.

This mixin is for the simplest purposes, it does not support mutiple color stops.

```
/* linear-gradient mixin by Jan Jerabek 2017 */
@mixin linear-gradient($top, $bottom, $deg: 0deg){
  background: $top;
  background: -moz-linear-gradient($deg,  $top 0%, $bottom 100%);
  background: -webkit-gradient(linear, $deg, left bottom,
              color-stop(0%,$top), color-stop(100%,$bottom));
  background: -webkit-linear-gradient(top,  $top 0%,$bottom 100%);
  background: -o-linear-gradient($deg,  $top 0%,$bottom 100%);
  background: -ms-linear-gradient($deg,  $top 0%,$bottom 100%);
  background: linear-gradient($deg,  $top 0%,$bottom 100%);
}
```

4.5.6 Transition

Transition mixin is used for basic animation, when switching between two values of one property. It will be used for menu sliding. Again, it takes all default transition properties, while generating all vendor prefixes for better browser compatibility.

If no attributes are explicitly provided, mixin will use transition for all properties, with 0.5s animation time and default easing.

```
/* Transition mixin by Jan Jerabek 2017 */
@mixin transition($property: all, $duration: 0.5s, $ease: ease){
  -webkit-transition: $property $duration $ease;
  -moz-transition: $property $duration $ease;
  -o-transition: $property $duration $ease;
  transition: $property $duration $ease;
}
```

4.5.7 Flexbox

Flexbox is a key part of responsive grid, which is the key function of the framework. To achieve optimal results and the best browser compatibility, the -ms- prefix was omitted, thus Internet Explorer is not supported up to the version 11.

However, older versions are supported partly, since the framework is mobile-first. There are two mixins that handle Flexbox - first is the substitute for display: flex; property, the second is for flex property.

```
/* display-flex mixin by Jan Jerabek 2017 */
@mixin display-flex {
  display: block;
  display: -webkit-box;
  display: -moz-box;
  display: -webkit-flex;
  display: flex;
}

/* flex mixin by Jan Jerabek 2017 */
@mixin flex($flex-grow: 0, $flex-shrink: 1, $flex-basis: auto) {
  -webkit-box-flex: $flex-grow $flex-shrink $flex-basis;
  -moz-box-flex: $flex-grow $flex-shrink $flex-basis;
  -webkit-flex: $flex-grow $flex-shrink $flex-basis;
  flex: $flex-grow $flex-shrink $flex-basis;
}
```

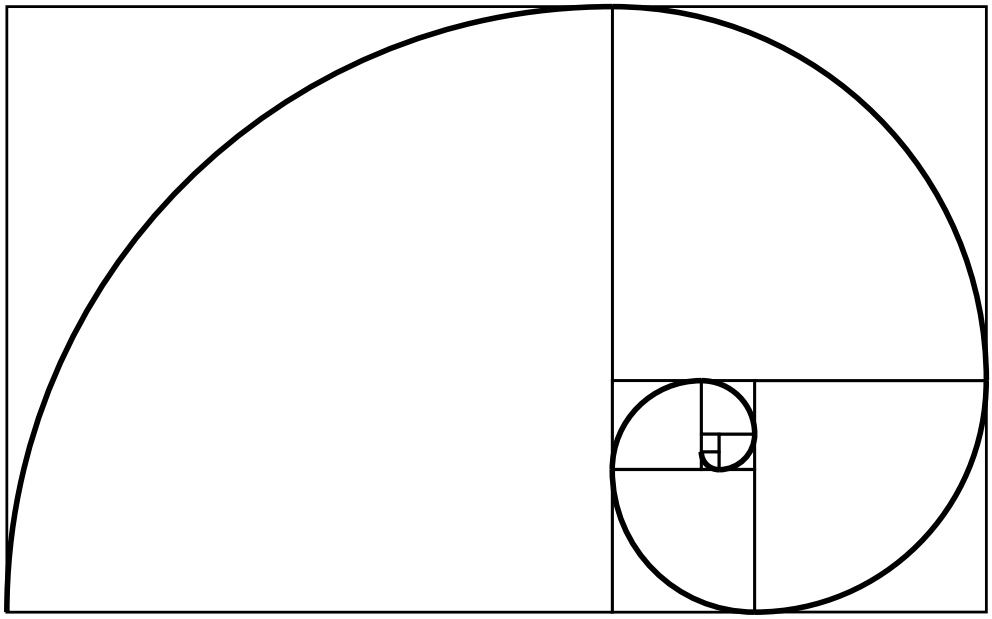
4.6 Typography

Typography is a key part of the Framework, because it determines how the text content is readable.

Responsiveness is a key to a well readable website. Using the breakpoints defined in responsive grid, the font-size is set to default 16px. For larger screens the size is increased to 18px. For smaller screens it is 12 - 14px, according to the device. Global font size is defined in pixels, as a reference size. All other font sizes are defined in EMs and REMs.

The line-height is set to so-called Golden ratio (1.61803399), which is a ratio of two numbers, that sum of the numbers is the same ratio to the largr of the numbers. This ratio is used in aesthetics (Architecture, Painting, Photography and Typography).

Figure 4.4: **Fibonacci's spiral of Golden ratio**



Source: Wikimedia Commons

The CSS default values are "reseted" using Normalize.css, which sets all the default CSS properties to be the same in all Internet Browsers. After resetting, the properties of the responsive typography are following:

```
$screen-xs: 600px;
$screen-sm: 900px;
$screen-md: 1200px;
$screen-lg: 1800px;


body {
  font-size: 16px;
  line-height: 1.61803399em;

  @media screen and (max-width: $screen-xs) {
    font-size: 12px;
  }
```

```
  @media screen and (max-width: $screen-sm) {
    font-size: 14px;
  }

  @media screen and (min-width: $screen-lg) {
    font-size: 18px;
  }

}
```

4.7 Deployment

In order to be even able to use the created Framework, it is necessary to compile it. There are three methods of compilation of this Framework:

- **Console**
- **IDE**
- **Webpack**

4.7.1 Console

To Compile the framework in console, which is useful for development, it is possible to achieve it using standard Terminal. The command to compile the framework to the output CSS is following:

```
sass input.scss styles.css --style compressed
```

This will create a file styles.css, minified (printd on one line). It is also possible to turn on the watch mode, which will start a file watcher, creating styles.css every time any file included to input.scss is changed until the user terminates the watcher:

```
sass --watch input.scss:styles.css --style compressed
```

4.7.2 IDE Plug-in

The second option of deployment of the CSS Framework is to let the editor save the minimized assets with all styles.

The most popular IDEs and Editors for web development include CSS compilers either native or as a plug-in. Every time the source file is saved (CMD+S / CTRL+S), the compilation script is triggered to immediately compile the output CSS file

Supported IDEs and Editors

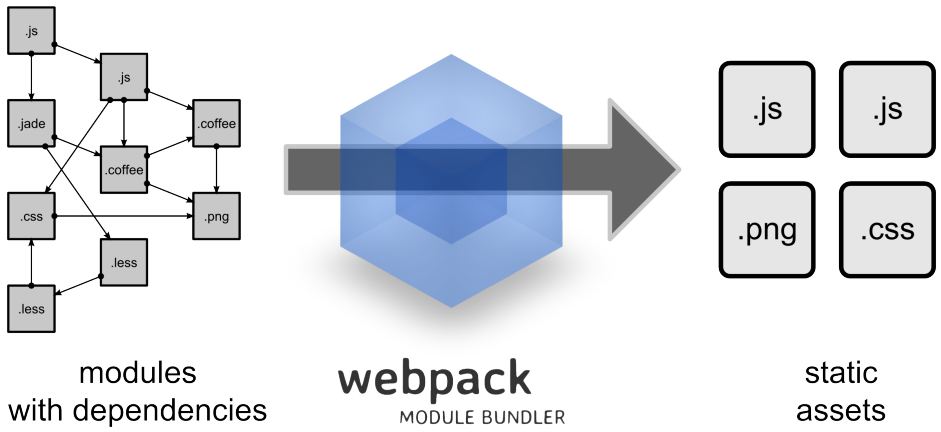
- Atom
- PhpStorm
- Netbeans
- Webstorm
- Sublime Text
- Koala
- CodeKit
- Prepros

4.7.3 Webpack

Webpack is more advanced module bundler with larger effect for JavaScript. It is, however, perfect for compilation of style sheets, since it can compile multiple languages (CSS / SCSS / LESS) and bundle them together to one minified CSS file for production environment, or debug-friendly code for development environment.

It also includes post-css processing - Autoprefixer, which will automatically add vendor prefixes wherever they miss in the project.

Figure 4.5: **What is Webpack?**



Source: http://webpack.github.io

```
const autoprefixer = require('autoprefixer')
const ExtractTextPlugin = require('extract-text-webpack-plugin')
const path = require('path')

const sassLoaders = [
  'css-loader',
  'postcss-loader',
  'sass-loader?indentedSyntax=sass&includePaths[]=' +
  path.resolve(__dirname, './src')
]

const config = {
  entry: {
    app: ['./src/index']
  },
  module: {
    loaders: [
      {
        test: /\.js$/,
        exclude: /node_modules/,
        loaders: ['babel-loader']
      },
```

```
      {
        test: /\.scss/,
        loader: ExtractTextPlugin.extract('style-loader',
        sassLoaders.join('!'))
      }
    ]
  },
  output: {
    filename: '[name].js',
    path: path.join(__dirname, './build'),
    publicPath: '/build'
  },
  plugins: [
    new ExtractTextPlugin('[name].css')
  ],
  postcss: [
    autoprefixer({
      browsers: ['last 2 versions']
    })
  ],
  resolve: {
    extensions: ['', '.js', '.scss'],
    root: [path.join(__dirname, './src')]
  }
}

module.exports = config
```

The basic config above defines the input and output folders a various filters for file processing. This config is the most basic way of configuring of Webpack.

Webpack can provide one time compilation as well as Dev-server, which runs as a process, watching all files that are bundled by Webpack for changes. Webpack does not recompile whole bundle every time that something is changed, but it rather provides small hot updates that are overwriting only the changes that were made. This benefits with a speed up in comparison to previous two ways of deployment.

4.8 Browser Compatibility

According to the browser analysis, in the Section 3.1.3, in 2017 it is not necessary to give up functionality in favor of better compatibility with more Internet browsers. For this reason, the framework focuses on modern technologies along with mobile devices - with consideration to performance.

Since the Framework is mobile-first, it brings backwards compatibility to the old browsers. Basic mobile version should be displayed correctly not only on mobiles, but on old computers as well. In this scope, there are no limits on compatibility with old browsers.

However, to achieve the best experience of the CSS framework, it has been tested on more than 150 Internet browsers using the online testing platform Browserling[5]. The frameworks takes advantage from Internet browser's auto-update feature and from drop of the official support of Internet Explorer.

Figure 4.6: **Browser compatibility**

Source: Own Drawing

After a deep analysis of the available statistics, testing and improving the code of the Framework in order to gain better compatibility, the compatibility is:

[5] www.browserling.com

- Microsoft Internet Explorer 11+
- Microsoft Edge (all)
- Google Chrome 51+
- Mozilla Firefox 26+
- Apple Safari 7+
- Opera 20+

Lower version do not necessarily mean lack of compatibility, but there are eg. known issues of behavior, that does not make 100% user experience.

Although, none of these issues are making the framework not to work. All versions that are compatible are dated back to 2014 (in average), making sufficient time amount for users to update (as mentioned before - browsers are usually provided with auto-update feature).

These platform-specific bugs are being fixed over the time and since they are representing minor issues, it was decided to not to handle with them, since fixing them would bring external JavaScript libraries that would not serve anything but a specific scenario under specific conditions.

Conclusion

The main objective of the thesis was to deliver an analysis of modern CSS development using CSS Framework or CSS precompilers. The theoretical part focuses on study of modern CSS Frameworks and Precompilers that bring much desired order to web development. Using either of them brings a basic backbone that makes code clean, minimized and most importantly - reusable. As of 2017, these two key elements have become a standard in web development.

The practical part, which directly follows the theoretical part focuses on design of a light-weight CSS Framework, that serves as an important support for a web developer.

The CSS Framework, as a part of the objective, was designed as a result of an analysis of existing solutions, with consideration to their weaknesses and outdatedness. The analysis was made on overall experience, grid systems and typography.

The scope of the CSS Framework was designed as a Flexbox responsive grid, set of Mixin components and typography settings. The Framework is fully customizable, reusable and supports wide selection of Internet browsers, that it has been tested on. The browser compatibility has been

narrowed down to browsers versioned to 2014. After an analysis of Internet browser usage development it was decided to omit old browsers, because their part in the distribution is minor.

The practical part also shows the options of deployment of a website using the CSS Framework, since certain components are using generated data, and they have to be recompiled each time, the website is being changed.

The next step of the CSS Framework development is to develop form components - easily editable and reusable components for web interaction. Another possible way of development is to integrate this CSS Framework to Symfony 3 templating system TWIG and to Javascript front-end framework React.

In final conclusion, this thesis gives an overview of web design methods, that are up to 2017, while giving up methods that are not necessary to use any more.

For the public, a modern CSS Framework has been developed to serve developers in responsive web design.

Personally, the thesis has given me a deep knowledge about responsive web design in cooperation with precompilers and CSS Frameworks.

Bibliography

[1] Bringhurst, R.: *The Elements of Typographic Style*. Hartley & Marks Publishers, second edition, 2002, ISBN 0881791326.

[2] Castro Elizabeth, H. B.: *HTML5 a CSS3: názorný průvodce tvorbou WWW stránek*. Computer Press, 2012, ISBN 9788025137338.

[3] COYIER, C.: *Sass vs. LESS*. [online] [cit. 2017-02-21]. Available at WWW: <`https://css-tricks.com/sass-vs-less`>

[4] Ducket, J.: *HTML & CSS - Design and Build Websites*. John Wiley & Sons, Inc., 2011, ISBN 9781118008188.

[5] Eccher, C.: *Profesionální webdesign: techniky a vzorová řešení pro XHTML a CSS*. Computer Press, 2010, ISBN 9788025126776.

[6] GALLAGHER, N.: *Normalize.css*. [online] [cit. 2017-02-22]. Available at WWW: <`https://necolas.github.io/normalize.css`>

[7] Gilbertson, D.: *The 100% correct way to do CSS breakpoints*. [online] [cit. 2017-02-17]. Available at WWW: <`https://medium.freecodecamp.com/the-100-correct-way-to-do-css-breakpoints-88d6a5ba1862`>

[8] Hampton Catlin, M. L. C.: *Pragmatic Guide to Sass*. The Pragmatic Bookshelf, 2011, ISBN 9781934356845.

[9] Kadlec, T.: *Responzivní design*. Zoner Press, 2014, ISBN 9788074132803.

[10] MEYER, E.: *CSS Tools: Reset CSS*. [online] [cit. 2017-02-22]. Available at WWW: <http://meyerweb.com/eric/tools/css/reset>

[11] Mills, C.: *Practical CSS3 Develop and Design*. Peachpit Press, 2013, ISBN 9780321823724.

[12] *Browser Display Statistics*. [online] [cit. 2017-02-01]. Available at WWW: <http://www.w3schools.com/browsers/browsers_display.asp>

[13] *CSS Compatibility in Internet Explorer*. [online] [cit. 2017-01-31]. Available at WWW: <https://msdn.microsoft.com/en-us/library/hh781508(v=vs.85).aspx>

[14] *CSS3 Browser Support Reference*. [online] [cit. 2017-01-31]. Available at WWW: <http://www.w3schools.com/cssref/css3_browsersupport.asp>

[15] *CSS3 Introduction*. [online] [cit. 2017-01-31]. Available at WWW: <http://www.w3schools.com/css/css3_intro.asp>

[16] *The Lengths of CSS*. [online] [cit. 2017-02-01]. Available at WWW: <https://css-tricks.com/the-lengths-of-css>

[17] *Microsoft will force Internet Explorer users to use latest version*. [online] [cit. 2017-01-31]. Available at WWW: <https://features.en.softonic.com/microsoft-finally-forces-internet-explorer-users-to-use-latest-version>

[18] *The Most Popular Browsers*. [online] [cit. 2017-01-31]. Available at WWW: <http://www.w3schools.com/browsers>

[19] *OS Platform Statistics*. [online] [cit. 2017-02-16]. Available at WWW: <`https://www.w3schools.com/browsers/browsers_os.asp`>

[20] *SASS Documentation*. [online] [cit. 2017-02-25]. Available at WWW: <`http://sass-lang.com/documentation/Sass/Script/Functions.html`>

[21] *Tim Berners-Lee*. [online] [cit. 2017-01-19]. Available at WWW: <`https://www.w3.org/People/Berners-Lee`>

[22] *Update Google Chrome*. [online] [cit. 2017-01-31]. Available at WWW:

List of Figures

List of Tables

Appendix A

List of abbreviations

CERN The European Organization for Nuclear Research

CM Centimeter

CSS Cascading Style Sheets

DEG Degrees

DOM Document Object Model

DRY Don't Repeat Yourself

EM "M"

HAML HTML Abstraction Markup Language

HTML HyperText Markup Language

IDE Integrated Development Environment

IE Internet Explorer

LESS Less CSS

MM Milimeters

OS Operating System

OOCSS Object-oriented CSS

PHP Hypertext Preprocessor

PX Pixel

REM Root Em

SASS Syntactically Awesome Stylesheets

UI User Interface

VH Viewport Height

VW Viewport Width

W3C World Wide Web Consortium

YAML YAML Ain't Markup Language

Appendix B

Normalize.css v5.0.0

```
/*! normalize.css v5.0.0 | MIT License |
github.com/necolas/normalize.css */

/**
 * 1. Change the default font family in all browsers
 *    (opinionated).
 * 2. Correct the line height in all browsers.
 * 3. Prevent adjustments of font size after
 *    orientation changes in IE on Windows Phone and in iOS.
 */

/* Document
   ========================================================== */

html {
  font-family: sans-serif; /* 1 */
  line-height: 1.15; /* 2 */
  -ms-text-size-adjust: 100%; /* 3 */
  -webkit-text-size-adjust: 100%; /* 3 */
}

/* Sections
   ========================================================== */

/**
 * Remove the margin in all browsers (opinionated).
```

```
 */

body {
  margin: 0;
}

/**
 * Add the correct display in IE 9-.
 */

article,
aside,
footer,
header,
nav,
section {
  display: block;
}

/**
 * Correct the font size and margin on `h1` elements
 * within `section` and
 * `article` contexts in Chrome, Firefox, and Safari.
 */

h1 {
  font-size: 2em;
  margin: 0.67em 0;
}

/* Grouping content
   ================================================== */

/**
 * Add the correct display in IE 9-.
 * 1. Add the correct display in IE.
 */

figcaption,
figure,
main { /* 1 */
```

```
  display: block;
}

/**
 * Add the correct margin in IE 8.
 */

figure {
  margin: 1em 40px;
}

/**
 * 1. Add the correct box sizing in Firefox.
 * 2. Show the overflow in Edge and IE.
 */

hr {
  box-sizing: content-box; /* 1 */
  height: 0; /* 1 */
  overflow: visible; /* 2 */
}

/**
 * 1. Correct the inheritance and scaling of
 *    font size in all browsers.
 * 2. Correct the odd `em` font sizing in all browsers.
 */

pre {
  font-family: monospace, monospace; /* 1 */
  font-size: 1em; /* 2 */
}

/* Text-level semantics
   ========================================================== */

/**
 * 1. Remove the gray background on active links in IE 10.
 * 2. Remove gaps in links underline in iOS 8+ and Safari 8+.
 */
```

```
a {
  background-color: transparent; /* 1 */
  -webkit-text-decoration-skip: objects; /* 2 */
}

/**
 * Remove the outline on focused links when
 * they are also active or hovered
 * in all browsers (opinionated).
 */

a:active,
a:hover {
  outline-width: 0;
}

/**
 * 1. Remove the bottom border in Firefox 39-.
 * 2. Add the correct text decoration in
 *    Chrome, Edge, IE, Opera, and Safari.
 */

abbr[title] {
  border-bottom: none; /* 1 */
  text-decoration: underline; /* 2 */
  text-decoration: underline dotted; /* 2 */
}

/**
 * Prevent the duplicate application of `bolder`
 * by the next rule in Safari 6.
 */

b,
strong {
  font-weight: inherit;
}

/**
 * Add the correct font weight in Chrome, Edge, and Safari.
 */
```

```
b,
strong {
  font-weight: bolder;
}

/**
 * 1. Correct the inheritance and scaling of
 *    font size in all browsers.
 * 2. Correct the odd ‘em‘ font sizing in all browsers.
 */

code,
kbd,
samp {
  font-family: monospace, monospace; /* 1 */
  font-size: 1em; /* 2 */
}

/**
 * Add the correct font style in Android 4.3-.
 */

dfn {
  font-style: italic;
}

/**
 * Add the correct background and color in IE 9-.
 */

mark {
  background-color: #ff0;
  color: #000;
}

/**
 * Add the correct font size in all browsers.
 */

small {
```

```
  font-size: 80%;
}

/**
 * Prevent `sub` and `sup` elements from
 * affecting the line height in
 * all browsers.
 */

sub,
sup {
  font-size: 75%;
  line-height: 0;
  position: relative;
  vertical-align: baseline;
}

sub {
  bottom: -0.25em;
}

sup {
  top: -0.5em;
}

/* Embedded content
   ================================================== */

/**
 * Add the correct display in IE 9-.
 */

audio,
video {
  display: inline-block;
}

/**
 * Add the correct display in iOS 4-7.
 */
```

```
audio:not([controls]) {
  display: none;
  height: 0;
}

/**
 * Remove the border on images inside links in IE 10-.
 */

img {
  border-style: none;
}

/**
 * Hide the overflow in IE.
 */

svg:not(:root) {
  overflow: hidden;
}

/* Forms
   ========================================================== */

/**
 * 1. Change the font styles in all browsers (opinionated).
 * 2. Remove the margin in Firefox and Safari.
 */

button,
input,
optgroup,
select,
textarea {
  font-family: sans-serif; /* 1 */
  font-size: 100%; /* 1 */
  line-height: 1.15; /* 1 */
  margin: 0; /* 2 */
}

/**
```

```
 * Show the overflow in IE.
 * 1. Show the overflow in Edge.
 */

button,
input { /* 1 */
  overflow: visible;
}

/**
 * Remove the inheritance of text transform
 * in Edge, Firefox, and IE.
 * 1. Remove the inheritance of text transform in Firefox.
 */

button,
select { /* 1 */
  text-transform: none;
}

/**
 * 1. Prevent a WebKit bug where (2)
 *    destroys native ‘audio‘ and ‘video‘
 *    controls in Android 4.
 * 2. Correct the inability to style
 *    clickable types in iOS and Safari.
 */

button,
html [type="button"], /* 1 */
[type="reset"],
[type="submit"] {
  -webkit-appearance: button; /* 2 */
}

/**
 * Remove the inner border and padding in Firefox.
 */

button::-moz-focus-inner,
[type="button"]::-moz-focus-inner,
```

```
[type="reset"]::-moz-focus-inner,
[type="submit"]::-moz-focus-inner {
  border-style: none;
  padding: 0;
}

/**
 * Restore the focus styles unset by the previous rule.
 */

button:-moz-focusring,
[type="button"]:-moz-focusring,
[type="reset"]:-moz-focusring,
[type="submit"]:-moz-focusring {
  outline: 1px dotted ButtonText;
}

/**
 * Change the border, margin, and padding
 * in all browsers (opinionated).
 */

fieldset {
  border: 1px solid #c0c0c0;
  margin: 0 2px;
  padding: 0.35em 0.625em 0.75em;
}

/**
 * 1. Correct the text wrapping in Edge and IE.
 * 2. Correct the color inheritance from
 *    `fieldset` elements in IE.
 * 3. Remove the padding so developers are
 *    not caught out when they zero out
 *    `fieldset` elements in all browsers.
 */

legend {
  box-sizing: border-box; /* 1 */
  color: inherit; /* 2 */
  display: table; /* 1 */
```

```
  max-width: 100%; /* 1 */
  padding: 0; /* 3 */
  white-space: normal; /* 1 */
}

/**
 * 1. Add the correct display in IE 9-.
 * 2. Add the correct vertical alignment in
 *    Chrome, Firefox, and Opera.
 */

progress {
  display: inline-block; /* 1 */
  vertical-align: baseline; /* 2 */
}

/**
 * Remove the default vertical scrollbar in IE.
 */

textarea {
  overflow: auto;
}

/**
 * 1. Add the correct box sizing in IE 10-.
 * 2. Remove the padding in IE 10-.
 */

[type="checkbox"],
[type="radio"] {
  box-sizing: border-box; /* 1 */
  padding: 0; /* 2 */
}

/**
 * Correct the cursor style of increment and
 * decrement buttons in Chrome.
 */

[type="number"]::-webkit-inner-spin-button,
```

```
[type="number"]::-webkit-outer-spin-button {
  height: auto;
}

/**
 * 1. Correct the odd appearance in Chrome and Safari.
 * 2. Correct the outline style in Safari.
 */

[type="search"] {
  -webkit-appearance: textfield; /* 1 */
  outline-offset: -2px; /* 2 */
}

/**
 * Remove the inner padding and cancel
 * buttons in Chrome and Safari on macOS.
 */

[type="search"]::-webkit-search-cancel-button,
[type="search"]::-webkit-search-decoration {
  -webkit-appearance: none;
}

/**
 * 1. Correct the inability to style
 *    clickable types in iOS and Safari.
 * 2. Change font properties to ‘inherit‘ in Safari.
 */

::-webkit-file-upload-button {
  -webkit-appearance: button; /* 1 */
  font: inherit; /* 2 */
}

/* Interactive
   ========================================================== */

/*
 * Add the correct display in IE 9-.
 * 1. Add the correct display in Edge, IE, and Firefox.
```

```
 */

details, /* 1 */
menu {
  display: block;
}

/*
 * Add the correct display in all browsers.
 */

summary {
  display: list-item;
}

/* Scripting
   ========================================================== */

/**
 * Add the correct display in IE 9-.
 */

canvas {
  display: inline-block;
}

/**
 * Add the correct display in IE.
 */

template {
  display: none;
}

/* Hidden
   ========================================================== */

/**
 * Add the correct display in IE 10-.
 */
```

```
[hidden] {
  display: none;
}
```