

Rogatnev Nikita

# Responsive Web Design

Bachelor's Thesis  
Information Technology

May 2015



MAMK

University of Applied Sciences

## DESCRIPTION

		<b>Date of the bachelor's thesis</b> 28.05.2015
<b>Author(s)</b> Rogatnev Nikita		<b>Degree programme and option</b> Technology, Communication and Transport
<b>Name of the bachelor's thesis</b> Responsive Web Design		
<b>Abstract</b> <p>In this thesis I described everything regarding to the responsive web design. First of all, I summed up and compared all the possible website layout approaches. The disadvantages of fixed, fluid, adaptive and hybrid layouts are so significant that it is undoubtedly clear that responsive approach is the best one.</p> <p>After that, in the theoretical part I defined what the responsive design was and described its main principles and techniques. To summarize, responsive design consists of the following: responsive layout with media queries to fit all possible devices, viewport and responsive typography and media.</p> <p>In the practical part I implemented one page responsive website without using frameworks or grid generators – just clean HTML5 and CSS3. In order to improve the complexity of the tasks multiple responsive media was included into the project. The project website includes responsive iFrame with a video from YouTube, responsive HTML5 audio player with music composition stored on the webserver and responsive icons provided by Font Awesome icon font. Typography is also adaptive, because it was set in “em” units, like it was stated in theoretical part of the study.</p>		
<b>Subject headings, (keywords)</b> HTML5, CSS3, RWD, responsive, web design		
<b>Pages</b> 67	<b>Language</b> English	<b>URN</b>
<b>Remarks, notes on appendices</b>		
<b>Tutor</b> Timo Mynttinen		<b>Employer of the bachelor's thesis</b>

## CONTENTS

1	INTRODUCTION .....	1
2	LAYOUT VARIATIONS .....	3
2.1	Fixed layout .....	4
2.2	Fluid layout .....	6
2.3	Adaptive layout .....	8
2.4	Hybrid layout .....	10
2.5	Conclusion .....	10
3	RESPONSIVE WEB DESIGN PRINCIPLES .....	12
3.1	Responsive layout .....	13
3.1.1	Responsive layout implementation option - Frameworks .....	15
3.1.2	Responsive layout implementation option - Layout generators .....	18
3.2	Responsive web design formula .....	19
3.3	Flexible margins and paddings .....	19
3.4	Viewport .....	21
3.5	Media queries .....	23
3.6	Responsive typography .....	24
3.7	Flexible media .....	26
3.7.1	Flexible images .....	27
3.7.2	Scalable Vector Graphics (SVG) .....	28
3.7.3	Icon fonts .....	29
3.7.4	SVG vs Icon Fonts .....	30
3.7.5	Flexible video .....	31
3.7.6	Flexible audio .....	32
3.8	Special cases .....	32
3.8.1	Touch optimization .....	32
3.8.2	Responsive mobile navigation .....	33
3.8.3	Old browsers' compatibility .....	35
4	RESPONSIVE WEB DESIGN IMPLEMENTATION .....	36
4.1	Requirement analysis .....	36
4.2	Workspace preparation .....	36
4.3	Project design .....	37
4.4	The layout building .....	38
4.5	CSS definition .....	42

4.6	Media query and viewport definition.....	48
4.7	Result screenshots.....	49
5	CONCLUSIONS.....	52
	BIBLIOGRAPHY.....	53
	APPENDIX 1. index.html file code.....	55
	APPENDIX 2. styles.css file code.....	58

## ABBREVIATIONS

WWW	World Wide Web
OS	Operating System
URL	Uniform Resource Locator
DOM	Document Object Model
CDN	Content Delivery Network
W3C	World Wide Web Consortium
RWD	Responsive Web Design
HTML	HyperText Markup Language
CSS	Cascading Style Sheets
SASS	Syntactically Awesome Style Sheets
JS	JavaScript
UI	User Interface
UX	User Experience
HD	High-Definition
FHD	Full High-Definition
Px	Pixel
Pt	Point (typography)
DPI	Dots Per Inch
IE	Internet Explorer
MIT	Massachusetts Institute of Technology
GPL	General Public License
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
SVG	Scalable Vector Graphics
PSD	Photoshop Document

## 1 INTRODUCTION

The web design industry is always in a constant motion. Web designing is continuously developed in case the invention of new technologies and tools. The main reason of a permanent design modification is the appearance of a huge range of new mobile devices, tablets and TVs with the access to the Internet. Injection of new web technologies, such as HTML5 and CSS3, is also significant. Web design plays a crucial role in creating of an effective website. This makes this research topic relevant.

Any work in the field of art begins with the selection of the canvas: an artist uses a sheet of paper or cloth, a sculptor chooses rock chunk. Irrespective of what an artist wants to make his first creative action is the choice of the canvas. Even before the first brush stroke or the first chisel hit the canvas sets the parameters and the shape, width and height of the future work and defines its boundaries.

Web designers try to follow this process. We even use the same word. We create a "canvas" - a blank document with a specific width, height, and shape parameters. However, there is a significant difference between sculptors and web designers or web developers: we are at a huge distance from the user and the user's browser window with all its individual peculiar inconsistencies and shortcomings. Let us face the truth: once the project is available online everything begins to depend on a man who looks at it – from the chosen font, color monitor, shape and size of the browser window.

Faced with such an uncertainty and flexibility we begin to set limitations: set the font size in pixels or create a fixed-width layout considering the minimum screen resolution. Setting such restrictions is a bit like choosing a canvas. These restrictions set strict parameters for future projects and give the stability that protects against the variability, which is initially inherent in the web.

However, there is a good and at the same time bad point in the World Wide Web. It ignores any restrictions. This applies to the parameters we set in our projects: they are too easy to get broken by browsers. If the width of the visitor web browser is slightly less than the expected minimum width, the user will face the fact that part of the content of a website will be cut or the user will be forced to use horizontal scrolling to view it.

Users can simply walk away from the viewing area, which causes wasting of user's time to find the way back, and the elimination of usability as a result.

Usability is a key factor in web design and in company marketing strategy nowadays (Taek-Hun Kim, 2007). Outward appearance of the website shows the face and status of the company and the quality of its services. Usability level shows the company's attitude to the user.

Website developers began making separate website designs for each device to prevent such a behavior of a layout in order to make website visitors feel comfortable while surfing through the websites. But, this approach is too time and money consuming. That is why, instead of creating a separate design for each new device or browser we could treat them as different manifestations of the same design. In other words, we have to create websites that are not only more flexible but also adaptable to the device display. The name of this approach is responsive web design, RWD. We can take advantage of the inherent web flexibility without abandoning the layout flow control. All we have to do to implement this technology in practice is to follow the web standards and change our attitude to web design.

The aim of this study is to summarize and analyze differences between fixed, fluid, hybrid and adaptive website design approaches in the theoretical part and to implement a responsive web design website for visual confirmation of the advantages of this methodology in the practical part.

In general, responsive web design consists of the following parts: responsive layout with flexible margins and paddings, media queries and flexible media. Section 3.1 consists of description and presentation of what responsive layout is and how it differs from the other ones. The differences between flexible and inflexible margins and paddings are described in Section 3.2. Section 3.4, in turn, contains description of how media could be flexible and why it is important.

## 2 LAYOUT VARIATIONS

Website page layout is a huge part of a web design study. It is the graphical component of website design which determines the arrangement of elements on a web page. Layout embraces not only the elements' position in the grid but also the UX. That fact makes the choice of a layout type a very important part of the UI design process. The main problem is to make the layout not only convenient for users, but also beautiful and attractive.

	Resolution	%
1	1366x768 HD	19.59%
2	1920x1080 16:9 HD 1080	13.25%
3	1024x768 4:3 X VGA	11.92%
4	1280x1024 5:4 SXGA	6.87%
5	1440x900 8:5 WSXGA	5.04%
6	1600x900 16:9 HD+ 900p	4.86%
7	1280x800 8:5 WXGA	4.71%
8	1680x1050 8:5 WSXGA+	3.45%
9	360x640	3.02%
10	768x1024	1.88%

TABLE 1. Screen resolution 2015 statistics (top 10) (<http://screenresolution.org/>)

Table 1 shows up-to-date users' screen resolution statistics. Screen resolution determines if content will be placed on the page correctly and completely. The prevailing 1024x768 for many years has lost its position. Website development trends changed as a logical consequence of that phenomenon. For the last few years UX increased its value. The fact that the browser window cuts a part of the content or that a horizontal scroll bar appears when the screen resolution is less than the website layout width become unacceptable.

In order to improve usability and conversion web designers began to invent different methodologies to make websites look fine at any device with any resolution. Here is a summary of layout variations mostly in chronological and by their popularity.

## 2.1 Fixed layout

Fixed or static layout is a page layout where the width of the content is rigidly set in pixels and does not change depending on the size of the browser window. Fixed layout behavior is shown in Figure 1.

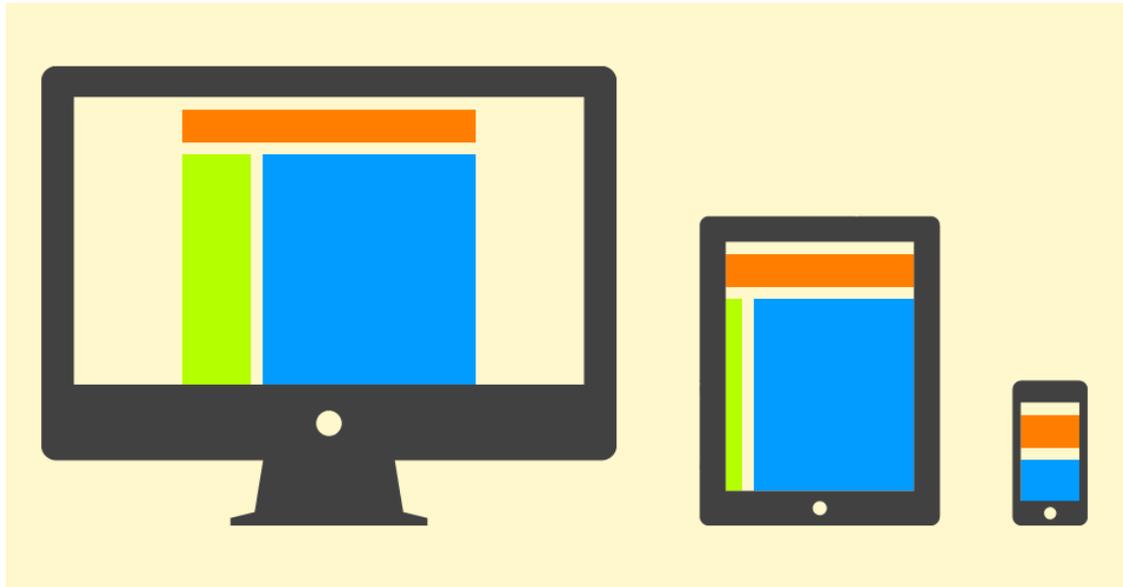


FIGURE 1. Fixed layout web page at various devices

Fixed layout is a relic of the past when the width of the page content was strictly dictated by the resolution of the most popular screens: 800x600 then 1024x768 and so on. Even nowadays it is really easy to find websites with a fixed layout width of 960 pixels. However, this way of making layouts became obsolete because of the following reasons:

- Fixed layout creates a lot of blank space for users with high-resolution displays, so that it breaks the "golden section", "rule of thirds" and other important principles of design.
- Low screen resolution causes the appearance of horizontal scrollbar.
- Seamless textures, patterns and big length images require large resolution.
- Fixed width layout is useless when it comes to usability.

Here is an example of a fixed layout website in Code 1 and a screenshot of the result in Figure 2.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <title>Example 1</title>
6      <style>
7          body {background: #2980b9; color: #FFF;
font-family: Helvetica; text-align: center; margin:
0;}
8          header, nav, section {border: 1px solid
rgba(255,255,255,0.8); margin-bottom: 10px; border-
radius: 3px;}
9          header {padding: 20px 0;}
10         nav, section {padding: 20px 0;}
11         .wrapper {width: 960px; margin: 0 auto;}
12         header {width: 960px;}
13         nav, section {float: left;}
14         nav {width: 200px; margin-right: 10px;}
15         section {width: 750px;}
16     </style>
17 </head>
18 <body>
19     <div class="wrapper">
20         <h1>Static Layout Example</h1>
21         <header>HEADER</header>
22         <nav>NAV</nav>
23         <section>SECTION</section>
24     </div>
25 </body>
26 </html>

```

CODE 1. Fixed layout page HTML5 and CSS3 code example

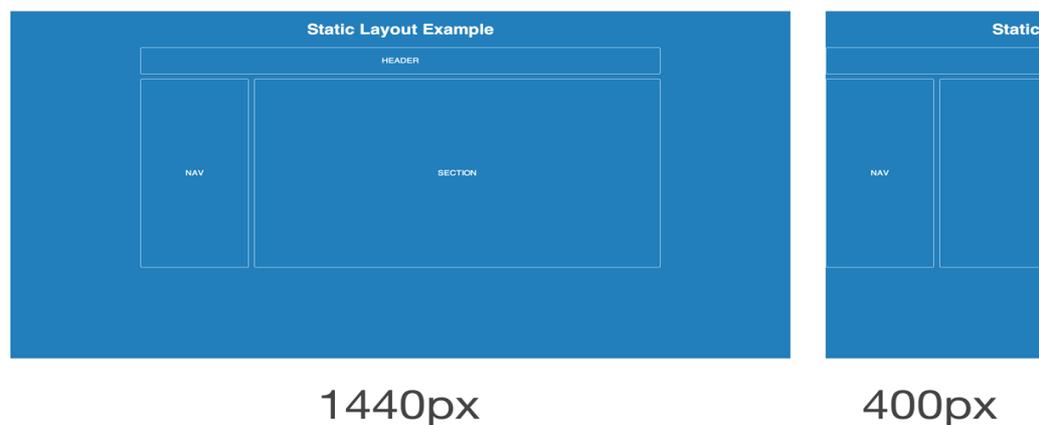


FIGURE 2. Fixed layout web page example in two browser window sizes

## 2.2 Fluid layout

In liquid or fluid layout the content width depends directly on any size of the browser window due to the use of structural elements set in relative indices, e.g. in percentage scale instead of static pixels. Fluid layout behavior is shown in Figure 3.

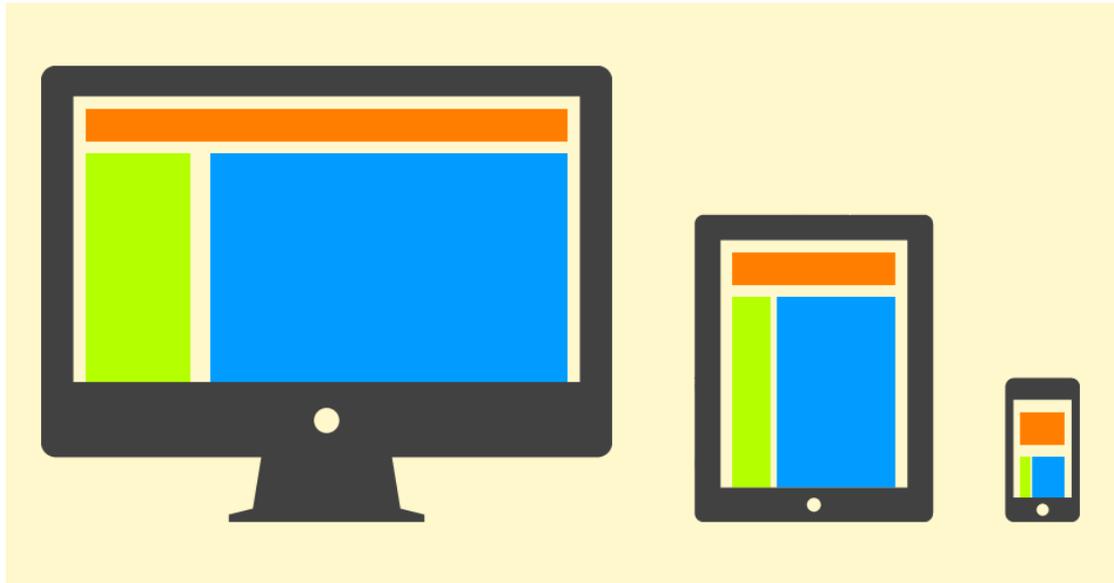


FIGURE 3. Fluid layout web page at various devices

This type of layout in its pure form is obsolete, too, as it only considers one type of devices and does not care about how the content will appear on critically large or small screens. The main disadvantages of fluid layout include the following:

- It is almost impossible to consider how it will look like on different resolutions when drawing the design.
- Large screen resolution causes creating long unreadable paragraph lines or/and a lot of free space.
- Graphical content elements must have multiple width properties to accommodate different screen resolutions.

Here is an example of a fluid layout website in Code 2 and a screenshot of the result in Figure 4.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <title>Example 2</title>
6      <style>
7          body {background: #2980b9; color: #FFF;
font-family: Helvetica; text-align: center; margin:
0;}
8          header, nav, section {border: 1px solid
rgba(255,255,255,0.8); margin-bottom: 10px; border-
radius: 3px;}
9          header {padding: 20px 0;}
10         nav, section {padding: 20px 0;}
11         .wrapper {width: 100%;}
12         nav, section {float: left;}
13         nav {width: 20.83333%; margin-right:
14 1.041667%;}
15         section {width: 78.125%;}
16     </style>
17 </head>
18 <body>
19     <div class="wrapper">
20         <h1>Static Layout Example</h1>
21         <header>HEADER</header>
22         <nav>NAV</nav>
23         <section>SECTION</section>
24     </div>
25 </body>
26 </html>

```

CODE 2. Fluid layout page HTML5 and CSS3 code example

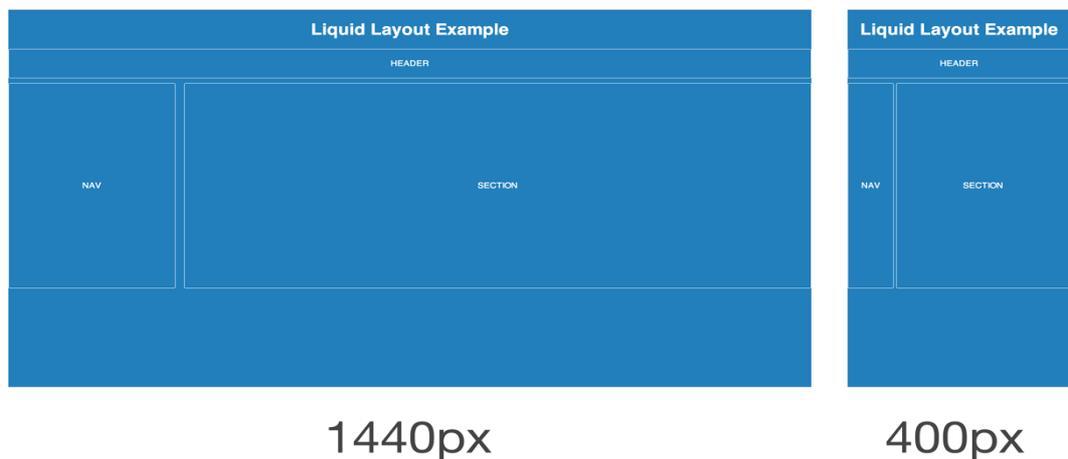


FIGURE 4. Fluid layout web page example in two browser window sizes

### 2.3 Adaptive layout

Adaptive layout is very close to the responsive layout type. It is based on the use of CSS media queries to adapt the content to different screens' resolutions and settings. Adaptive layout behavior is shown in Figure 5.

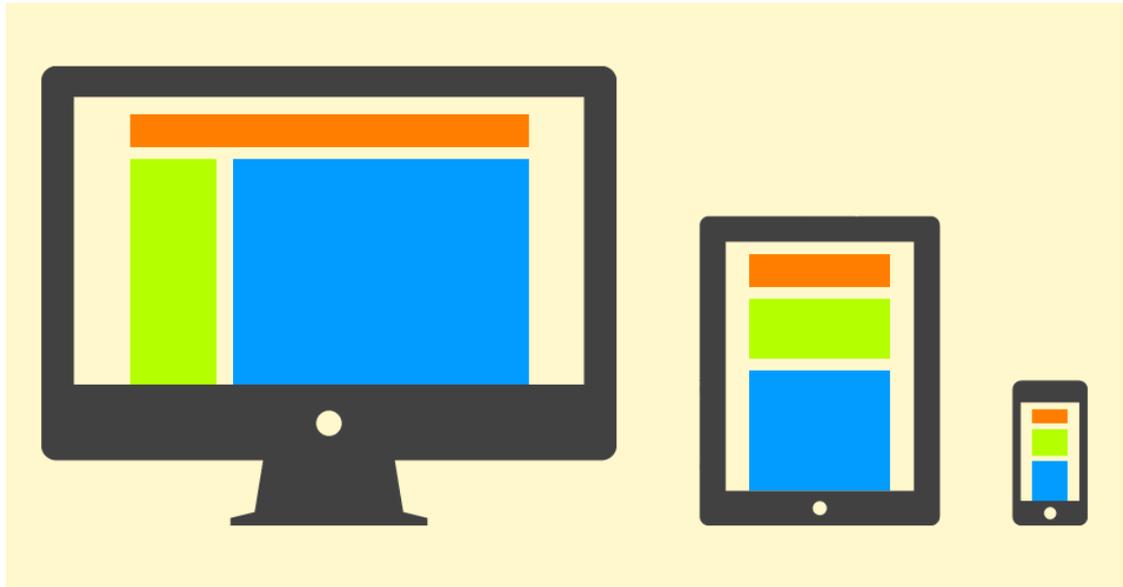


FIGURE 5. Adaptive layout web page at various devices

Media queries are part of the CSS3 specification which help to clarify the scope of the CSS selector. It is a specifying parameter unit of the output device, such as the type, width and height of the browser window, resolution and the orientation on canvas.

The main difference between the adaptive and responsive types is that in adaptive layouts the page "jumps" shifting and adapting content every time at control points. That is, the media requests the content to describe fixed positions for each of the control points. As a result, we have a set of multiple fixed layouts for different screen resolutions.

The disadvantage of this approach for the layout of the pages is obvious - we cannot predict how the content blocks will look on all user devices, because the distance between reference points could be quite large. This approach is particularly relevant, if the critical points are not about the width of the most common devices, but due to the web page design.

Here is an example of an adaptive layout website in Code 3 and a screenshot of the result in Figure 6.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <title>Example 3</title>
6      <style>
7          body {background: #2980b9; color: #FFF;
font-family: Helvetica; text-align: center; margin:
0;}
8          header, nav, section {border: 1px solid
rgba(255,255,255,0.8); margin-bottom: 10px; border-
radius: 3px;}
9          header {padding: 20px 0;}
10         nav, section {padding: 20px 0;}
11         .wrapper {width: 400px; margin: 0 auto;}
12         header {width: 400px;}
13         nav {width: 400px;}
14         section {width: 400px;}
15         @media screen and (min-width: 800px) {
16             .wrapper {width: 640px;}
17             header {width: 640px;}
18             nav, section {float: left;}
19             nav {width: 133px; margin-right: 10px;}
20             section {width: 497px;}
21         }
22         @media screen and (min-width: 1000px) {
23             .wrapper {width: 960px;}
24             header {width: 960px;}
25             nav {width: 200px;}
26             section {width: 750px;}
27         }
28     </style>
29 </head>
30 <body>
31     <div class="wrapper">
32         <h1>Static Layout Example</h1>
33         <header>HEADER</header>
34         <nav>NAV</nav>
35         <section>SECTION</section>
36     </div>
37 </body>
38 </html>

```

CODE 3. Adaptive layout page HTML5 and CSS3 code example

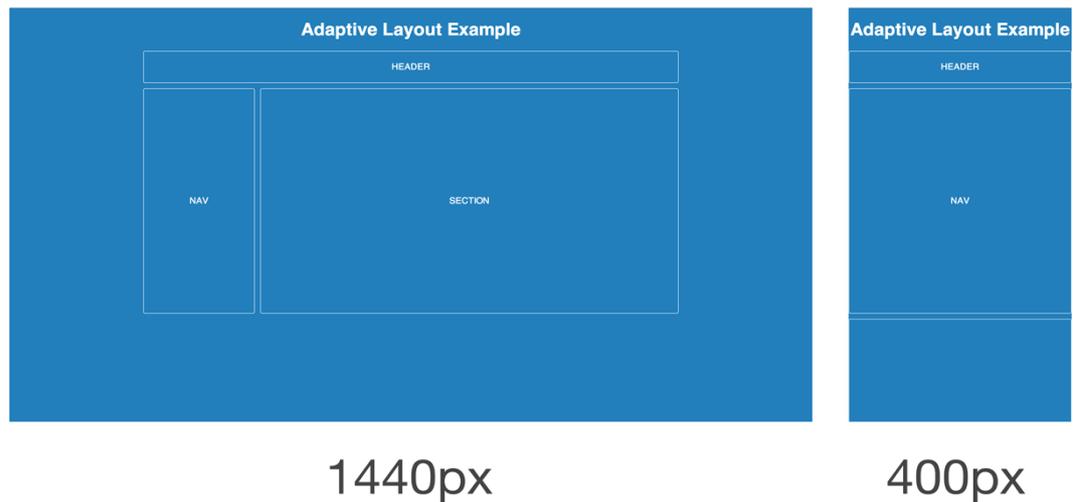


FIGURE 6. Adaptive layout web page example in two browser window sizes

## 2.4 Hybrid layout

There is no reason to deny the fact that there are other solutions for creating website layouts. Layout adaptation for different devices can be something between the adaptive and the responsive layout. This situation often occurs as an attempt to optimize the existing site for different types of devices (“mobile last”).

There are no specific principles for the hybrid layout, that is why there is no reason to consider it as a complete type of a layout design approach.

## 2.5 Conclusion

I have described most of the possible website layout options. But all of them have a lot of shortcomings. That fact provoked a new approach of creating website layouts and web design in general. What is then needed for creating a responsive web design? If we talk about the development of the layout of the page, we need three main components:

- Flexible grid-based layout
- Flexible images
- Media queries (module specification CSS3)

In the following paragraphs I will describe these components and explain how each of them makes this web design approach responsive. In the practical part I will create a responsive website which is able to adapt to the user's browser or device limitations. That will be a website that almost entirely meets the needs of the user.

### 3 RESPONSIVE WEB DESIGN PRINCIPLES

Responsive web design or RWD is a web design creation approach in which a website is developed with the expectation of providing easy and user-friendly design, including easy viewing of the web page with a minimum resizing and extra spins on a wide range of devices. The responsive design involves the following features:

- While developing responsive layouts only clean HTML 4/5 and CSS3 could be used without a connection of a JavaScript or some JS framework in order to determine the elements' responsiveness.
- Responsive layout determines how the website elements look on different devices. However, these elements should not be hidden or replaced by other ones and their behavior as well as their functions should not be changed.
- RWD should have an adaptive layout. This point is explained and demonstrated in section 3.1.
- RWD does not intend to work with the Domain Object Model, DOM, change the hierarchy or nesting of blocks and objects while changing the markup type.
- The CSS3 media queries module should be used in order to specify different styles or style sheets depending on the screen resolution, size and other device of browser characteristics.
- All elements should be located within the modular grid.
- All content should be flexible – depending on the size of the screen without quality lost.

#### 3.1 Responsive layout

The difference between adaptive and responsive methodologies comes down to how the site changes between breakpoints; adaptive is essentially a series of fixed-width layouts, whereas responsive uses flexible dimensions so even between breakpoints sites have fluidity. (Gasston, 2013)

Responsive layout is based on the principle of "rubber", but also uses media queries to make the content fit the width of the device. Such an approach does not allow content to make adaptive approach "jumps" at control points and therefore changes are smooth between break points. Responsive layout behavior is shown in Figure 7 and Figure 8.

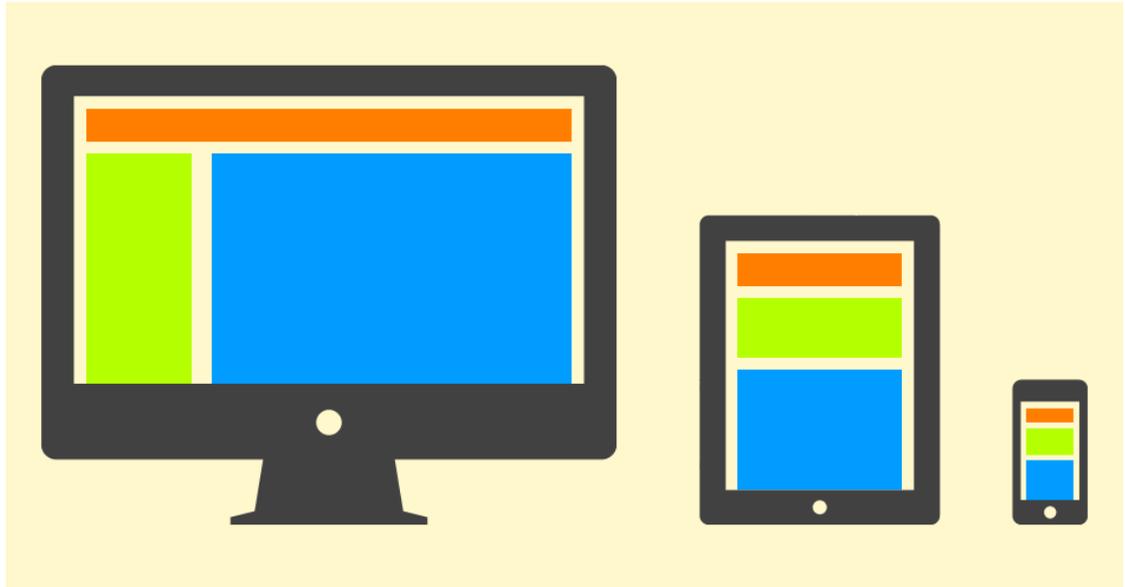


FIGURE 7. Responsive layout web page at various devices

To summarize, responsive approach has all the advantages of the other layout types, but also fixes their disadvantages. That makes responsive layout the most advanced and modern approach nowadays.

Here is an example of a responsive layout website in Code 4 and a screenshot of the result in Figure 8.

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <title>Example 2</title>
6      <style>
7          body {background: #2980b9; color: #FFF;
font-family: Helvetica; text-align: center; margin:
0;}
8          header, nav, section {border: 1px solid
rgba(255,255,255,0.8); margin-bottom: 10px; border-
radius: 3px;}
9          header {padding: 20px 0;}
10         nav, section {padding: 20px 0;}
11         .wrapper {max-width: 100%; margin: 0 auto;}

```

```

12     @media screen and (min-width: 800px) {
13         .wrapper {max-width: 90%;}
14         header {width: 100%;}
15         nav, section {float: left;}
16         nav {width: 20.83333%; margin-right:
17 1.041667%;}
18         section {width: 78.125%;}
19     }
20     @media screen and (min-width: 1000px) {
21         .wrapper {max-width: 66.66667%;}
22     }
23 </style>
24 </head>
25 <body>
26     <div class="wrapper">
27         <h1>Static Layout Example</h1>
28         <header>HEADER</header>
29         <nav>NAV</nav>
30         <section>SECTION</section>
31     </div>
32 </body>
33 </html>

```

CODE 4. Responsive layout page HTML5 and CSS3 code example

Responsive layout Code 4 is approximately similar to the adaptive layout Code 3, but the elements' dimensions are set in percentages, not in pixels. Therefore, the behavior of a layout becomes smooth while resizing the window, eliminating the "jumps" of the adaptive approach. This fact makes this layout type the most advanced and responsive nowadays.

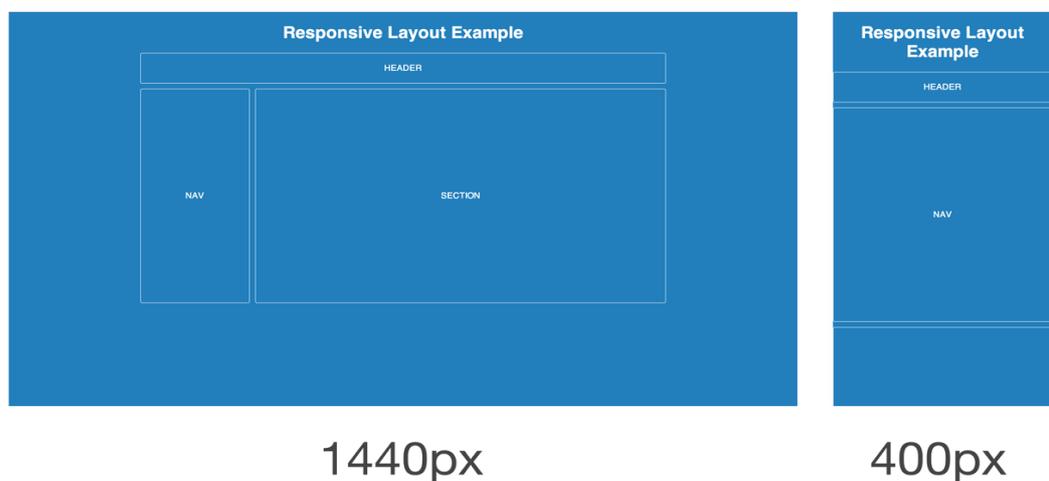


FIGURE 8. Responsive layout web page example in two browser window sizes

Multiple layout generators and frameworks have been developed in order to reduce the layout development time. These methods are briefly described in Sections 3.1.1 and 3.1.2. However, only pure HTML5 and CSS3 are used in the practical implementation part in order to present how responsive layout works and what exactly it consists of.

### 3.1.1 Responsive layout implementation option - Frameworks

The permanent creation and development of dynamic cross-browser websites and web applications requires going beyond the standard techniques. Fortunately, there are several frameworks that provide the functionality needed for developers to increase efficiency and opportunities to save valuable time.

Framework is a software platform that defines the structure of a software system; software that facilitates the development and integration of the various components of a large software project. In the context of web development CSS frameworks are most useful.

There are plenty of CSS frameworks in the form of libraries of media queries. The usage of frameworks involves the following features:

- Frameworks allow great reduction of development time because most of the code and rules are already written.
- Frameworks give a lot of unused and unnecessary CSS selectors.
- Studying frameworks' code greatly improves front-end skills.

Figure 9 shows the top five best CSS3 frameworks.

<a href="#">Bootstrap</a>	★ 79k+	LESS и SASS	IE8+	<a href="#">MIT</a>
<a href="#">Foundation</a>	★ 19k+	SASS	IE9+	<a href="#">MIT</a>
<a href="#">Semantic-UI</a>	★ 15k+	LESS	IE9+	<a href="#">MIT</a>
<a href="#">Skeleton</a>	★ 8k+	-	IE9+	<a href="#">MIT</a>
<a href="#">UIkit</a>	★ 4,4k+	LESS	IE7+	<a href="#">MIT</a>

FIGURE 9. The best top five CSS3 frameworks (<http://usablica.github.io/front-end-frameworks/>)

In most of the cases they are based on a grid of twelve dividing cells. The only steps required for the start of using the framework in a project are: assign necessary classes to the elements corresponding to the block size and connect the framework itself.

Twitter Bootstrap is one of the most progressive and widely used CSS frameworks. Let's consider a bit of its structure and the method of use.

Bootstrap uses four modular grids:

- XS - works always.
- SM - works when viewport  $\geq$  @screen-sm (768px by default).
- MD - works when viewport  $\geq$  @screen-md (992px by default).
- LG - works when viewport  $\geq$  @screen-lg (1200px by default).

An example of this method can be seen in Figure 10 and in Code 5.

```

1  <div class="container">
2    <div class="jumbotron">
3      <h1>My First Bootstrap Page</h1>
4      <p>Resize this responsive page to see the ef-
5  fect!</p>
6    </div>
7    <div class="row">
8      <div class="col-sm-4">
9        <h3>Column 1</h3>
10       <p>Lorem ipsum dolor sit amet, consectetur
11  adipiscing elit...</p>
12       <p>Ut enim ad minim veniam, quis nostrud ex-
13  ercitation ullamco laboris...</p>
14     </div>
15     <div class="col-sm-4">
16       <h3>Column 2</h3>
17       <p>Lorem ipsum dolor sit amet, consectetur
18  adipiscing elit...</p>
19       <p>Ut enim ad minim veniam, quis nostrud ex-
20  ercitation ullamco laboris...</p>
21     </div>
22     <div class="col-sm-4">
23       <h3>Column 3</h3>
24       <p>Lorem ipsum dolor sit amet, consectetur
25  adipiscing elit...</p>
26       <p>Ut enim ad minim veniam, quis nostrud ex-
27  ercitation ullamco laboris...</p>

```

```
28     </div>
29 </div>
30 </div>
```

### CODE 5. Bootstrap page markup example

Twitter Bootstrap uses a 12-column grid system. Thus, its span values run from 1 through 12. `span12` is full-width, `span8` is two-thirds, `span6` is half width, and so on. The `div class="row"` is built to contain a row of columns. Each new row creates a new zone for laying out columns as desired. (Cochran, 2012)

There are three columns in the Bootstrap Code 5 example. They are stated with the `col-sm-4` class. Figure 10 and Figure 11 show how this layout looks on normal and mobile browsers. The maximum width of a grid is 12, so there are three equal columns, 4/12 each.

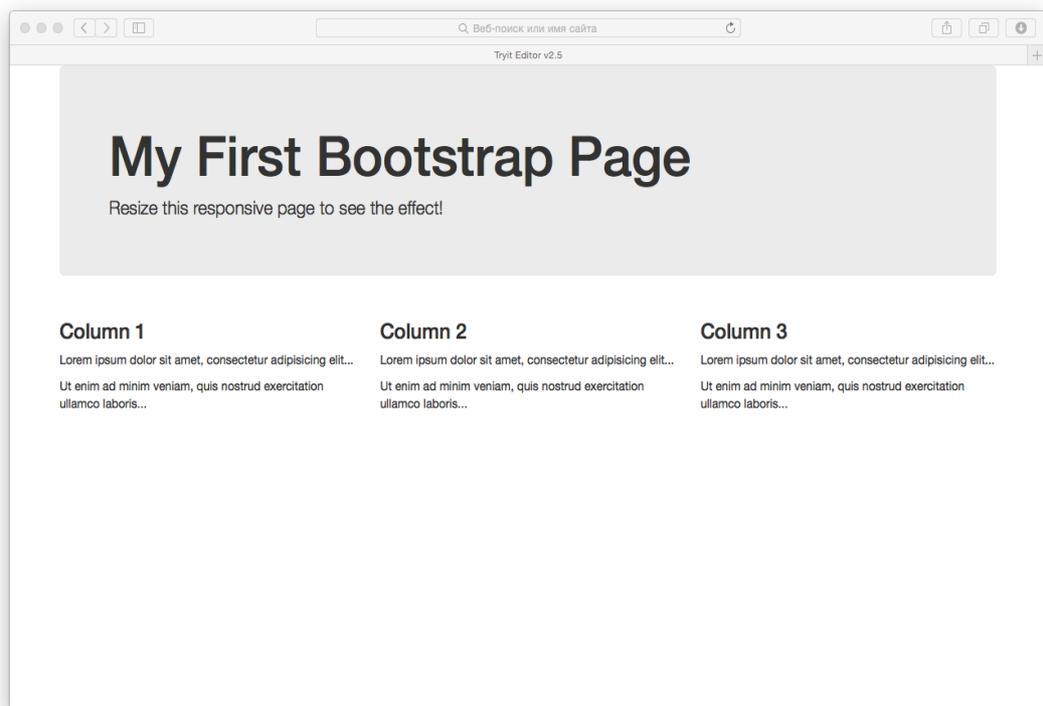


FIGURE 10. Bootstrap page example on a standard PC screen

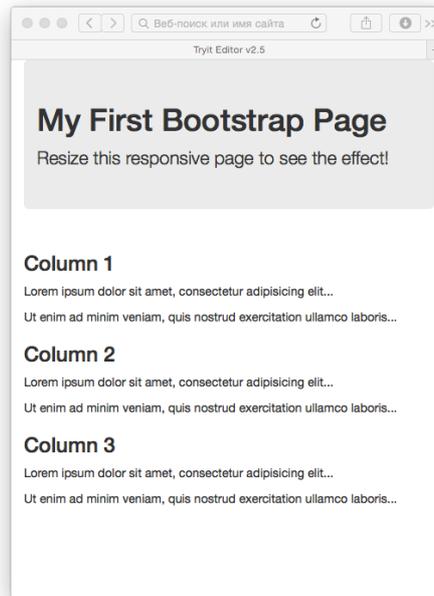


FIGURE 11. Bootstrap page example on a mobile device

### 3.1.2 Responsive layout implementation option - Layout generators

The most unpredictable and limited approach is the use of the responsive layout or the grid generator. There is a grid generator example in Figure 12. Such a method is the worst because it is not always clear what is happening in a particular case and the generated result is almost impossible to administer and modify.

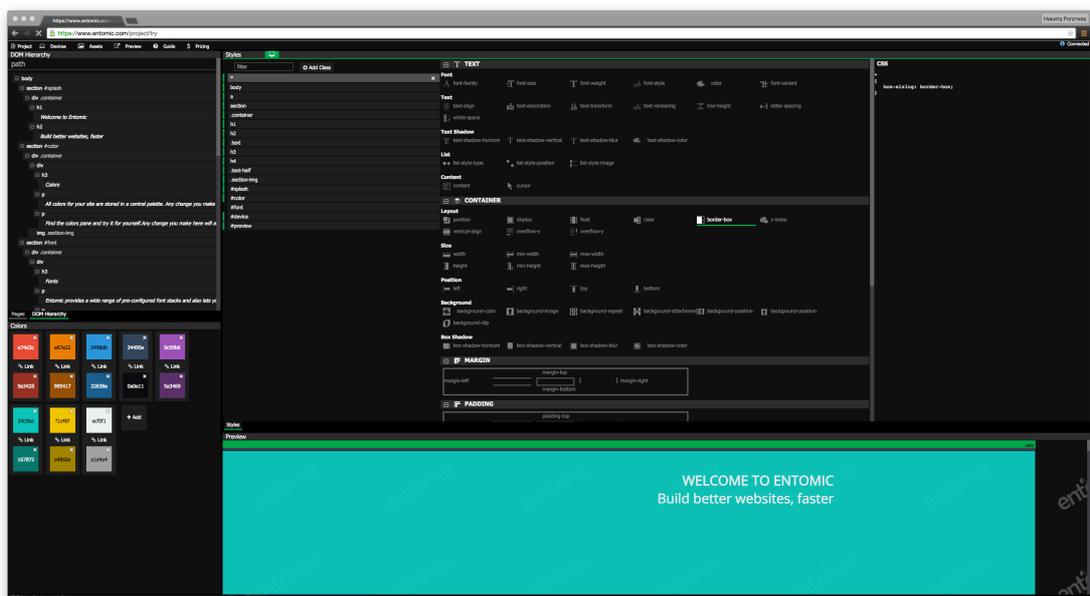


FIGURE 12. “Entomic” responsive layout generator

In general, this method is suitable only for beginners or uneducated developers who do not want to spend their time on learning the native technology. But the use of grid generators is suitable when a project should be quickly launched into production.

### 3.2 Responsive web design formula

There is a formula to find out the proportions of a flexible layout, margins and paddings using relative values. In order to find out the relative width of a target element the target width should be divided by the width of its parent element.

$$\text{Target} / \text{Context} = \text{Result}$$

#### FORMULA 1. Responsive web design formula

For example, there is a PSD with 1000px width layout design. It has two parts: 250px on the left and 750px on the right. I typeset them like in Code 6.

```

1  .left_column {
2      width: 25%; /* 250px / 1000px = 0,25 */
3      float: left;
4  }
5
6  .right_column {
7      width: 75%; /* 750px / 1000px = 0,75 */
8      float: right;
9  }
```

#### CODE 6. Two responsive blocks CSS example

This formula will repeatedly occur throughout this thesis. Most of the responsive web design calculations are made with its help.

### 3.3 Flexible margins and paddings

While creating a responsive layout we used to set all values in percentages. Therefore, if we define margins in pixels, there will be unwanted empty spaces. In order to avoid this, paddings and margins should be also sated with a percentage. However, it should be noted that the contexts for flexible fields and flexible paddings differ a bit. Here are the rules to follow in the context of flexible margins and paddings:

- When specifying flexible paddings for an element, the width of the container element should be considered as a context.
- When specifying flexible field for the element, the width of the element should be considered as a context.

Viewport height and width always change from device to device. Flexible margins and paddings should be used in the responsive website layout. In order to make responsive margins and paddings Formula 1 should be used.

Here is a simple flexible margin example in Code 7. There is an unordered list with a left margin set to 25px in a 150px width container. The existing CSS is shown in Code 7.

```

1  .container {
2      width: 150px;
3  }
4  .container ul {
5      margin-left: 25px;
6  }
```

#### CODE 7. Not responsive CSS margin example

I should divide to make this margin responsive. The updated CSS is shown in Code 8.

```

1  .container {
2      width: 150px;
3  }
4  .container ul {
5      margin-left: 16.666667%;
6      /* 25px / 150px * 100 = 16.666667 */
7  }
```

#### CODE 8. Responsive CSS margin example

There is only one difference when setting the responsive paddings compared to the responsive margins. The width of the element itself should be used in Formula 1. For example, there is a 300px width paragraph with 5px 10px 0 paddings

```

1  /* Not responsive padding */
2  .container p {
3      width: 300px;
```

```

4     padding: 5px 10px 0;
5   }
6
7   /* Responsive padding */
8   .container p {
9     width: 300px;
10    padding: 5px 3.33333% 0;
11    /* 10px / 300px * 100 = 3.33333 */
12  }

```

CODE 9. Responsive and not responsive CSS padding examples

### 3.4 Viewport

A viewport is a visible area of the page in the browser, the browser window minus the scrollbar. An `<HTML>` tag and relative element dimensions are calculated regarding the size of the viewport.

Since the mobile devices' screen sizes are small, the implementation of the viewport should be different from the desktop. Otherwise, it may cause some layout problems like it is shown in Figure 13. This problem occurs because there are two types of pixels: device pixels and CSS pixels which are a bit less stable.

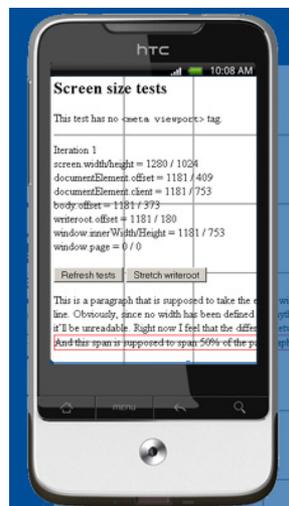


FIGURE 13. Viewport content cut example (<http://www.quirksmode.org/>)

In order to understand how to properly use viewport two new concepts should be introduced: layout viewport and visual viewport. The layout viewport is similar to device pixels in that its measurements are always the same, regardless of the orientation or

zoom level. The visual viewport, however, varies. This is the part of the page that's actually shown on the screen at any given point. (Kadlec, 2013) Visual viewport example could be found in Figure 14.



FIGURE 14. Visual viewport example (<http://www.quirksmode.org/>)

In order to control such a behavior the viewport meta tag should be used. Viewport meta tags are placed in the head tag of a HTML document and consist of the following parts:

- `device-width` value sets the width of the screen in CSS pixels. The values available are from 200 to 10000 or the "device-width".
- `device-height` value sets the height of the screen in CSS pixels. The values available are from 223 to 10000 or the "device-height".
- `initial-scale` property controls the first page load zoom level. The values available are from 0.1 to 10.
- `minimum-scale`, `maximum-scale` and `user-scalable` properties control how users could zoom in or out. The values available are from 0.1 to 10.
- Zooming functionality could be turned on or turned off by setting the `user-scalable` property. The values available are "yes" or "no".

Code 10 demonstrates a media query example:

```
1 <meta
```

```

name="viewport"
content="width=device-width,
        initial-scale=1.0,
        minimum-scale=1.0,
        maximum-scale=1.0,
        user-scalable=no" >

```

CODE 10. Viewport meta tag example

### 3.5 Media queries

Most commonly, media queries are used to create a responsive design, where different CSS styling is applied to different screen sizes. This lets designers craft sites that are pleasant to use and easy to read – even on very small screens – without creating completely separate mobile websites or themes. (WordPress.com, 2015)

The syntax of media queries allows the creation of rules that are applied on the basis of characteristics of the device. The general form of a media query is shown in Code 11.

```

1  @media [not|only] type [and] (expr) {
2      /* CSS rules used when query matches */
3      rules
4  }

```

CODE 11. Media query structure

While there are several different items we can query on, the ones used most often for responsive web design are min-width, max-width, min-height and max-height. Table 2 shows each media query attribute and its result.

Attribute	Result
min-width	Rules applied for any browser width over the value defined in the query.
max-width	Rules applied for any browser width below the value defined in the query.
min-height	Rules applied for any browser height over the value defined in the query.
max-height	Rules applied for any browser height below the value defined in the query.
orientation=portrait	Rules applied for any browser where the height is greater than or equal to the width.

<code>orientation=landscape</code>	Rules for any browser where the width is greater than the height.
------------------------------------	---

TABLE 2. Media queries attributes and results table (<https://developers.google.com/>)

As a result, each media query has four basic components: specification of a media type of a device to the target, media expressions, logical keyword such as “and”, “or”, “not” or “only” and CSS rules used when query matches.

The value in rounded brackets is a breakpoint. Breakpoints are often described as the points at which CSS media queries are activated and thus when changes to the page styles are applied. (Hay, 2013)

Code 12 shows multiple media queries examples for different user’s window widths. CSS rules for each width could be added inside the brackets “{}”.

```

1  @media only screen and (min-width: 480px) {}
2  @media only screen and (min-width: 768px) {}
3  @media only screen and (min-width: 992px) {}
4  @media only screen and (min-width: 1382px) {}

```

CODE 12. Media queries example

### 3.6 Responsive typography

The main objective of responsive typography is to make the text more readable on different devices. The devices differ in terms of use. And the most important point here is the distance from the screen to the users’ eye. The more this distance is, the larger text should be.

One of the most confusing aspects in the use of CSS is a “font-size” property for scaling text. We can change the browser text size by using CSS with four different units of measurement. Which of these four units is the best suited for the web? This is an issue that has generated a variety of discussions and criticism. Here is the list of the most commonly used text size units of measurement:

- “Px” are the fixed size units that are used, for example, to read some text information on a computer screen. A pixel is a single point on the computer monitor,

the smallest element of the screen resolution. A lot of web designers use "px" in web documents in order to obtain a pixel-perfect representation of the website displayed in the browser. One of the problems with the use of "px" is that these units do not allow changing the scale for visually impaired readers or mobile devices.

- “Em” is a scalable unit that is used in web documents. It is equal to the current font-size. For example, if the page font-size is 12pt, the 1em is equal to 12pt, 2em will be equal to 24pt, etc. The use of “em” becomes more popular in web due to scalability and the possibility to be used in mobile devices.
- “Pt” unit is traditionally used in print media. It is equal to 1/72 of an inch. “Pt” as well as “px” have a fixed unit size and therefore they cannot be scaled.
- “%” units of measurement are similar to “em” except for a few fundamental differences. Figure 15 shows what happens when «1em» is a basic unit and when the user changes the font size in browser settings.

body = 1em	Text-Size: "Smallest"	Text-Size: "Largest"
font-size: 1em	The quick brown fox jumps over the lazy dog.	The quick The quick br
font-size: 100%	The quick brown fox jumps over the lazy dog.	The quick br

FIGURE 15 Font size when user changes the text size in the browser

When the client browser text size is set to the medium size, there is no difference between "em" and "%". However, when the parameter got changed the difference becomes very huge. When the smallest size is set, "em" is smaller than "%". Similarly, "em" appears to be much larger than the "%" in case of the largest size. "Em" scales too sharply - the smallest text becomes illegible on some machines.

Theoretically, “em” unit is a new and upcoming standard font size, but in practice "%" units allow users to display text in a more consistent and convenient way. The "%" units are changing client text settings within a reasonable proportions. This fact allows designers to maintain readability, accessibility and design. Therefore, the choice between "em" and "%" as a responsive font size unit remains up to the developers' habit.

### 3.7 Flexible media

HyperText Markup Language pages and Cascading Style Sheets load very quick. But web content does not only consist out of text information. There are multiple media uploaded onto the WWW in a great amount every day. For example, more then 300 million new photos are added every day to Facebook (Pingdom AB, 2013).

Approximately every website uses some media. A huge amount of media makes the webpages download slowly, but the bandwidth is still a big issue. That is why, all media should be compressed and have a defined scale, but without the quality loss. In the following sections I will determine what flexible media should look like and what the best ways of connecting them to the website are.

Almost all smart phones are equipped with a screen with a high DPI today. The most advanced have a FHD resolution already, which is still not the most popular resolution on the desktop screens, by the way. Reading texts on such screens in a great pleasure because the borders are not sharp. But, icons and images look blurry when being increased to the same physical size as the monitor. This problem occurs not only on mobile devices, notebooks and desktop computers are also getting massively displays with high density points.

Therefore how could this problem with pictures be solved? The first option is to use larger photos than needed for a "standard" monitor. Then, on the FHD screens the photo will be displayed in the 1 to 1 scale and it will be smaller at "standard". For example, all the graphics files on the official Apple website are exactly twice wider and higher than necessary for normal computer screens. Of course, this approach must be used with caution, since it increases the page "weight" and the load on the server fourfold. However, for promotional pictures, in the case of Apple, this may be fully justified.

Another solution which should definitely be adopted is the rejection of raster graphics wherever it is possible. All modern browsers, including the mobile one support SVG. This means that it is allowed to embed vector objects directly into Web pages. In such a way, users will get not only high-quality graphics, but also an enormous traffic saving, as vector "weigh" is much smaller than raster graphics.

### 3.7.1 Flexible images

One of the most important points of RWD is the ability of images to change their size depending on the device and browser window size. By default the width and height of images is set by the use of pixels. That way of scaling is inflexible, because it fixes the image size. It is easy to implement a flexible image with percentages by keeping them from being wider than the parent element. Code 13 shows how the `max-width` property should be used.

```
1  img {
2      max-width: 100%;
3      height: auto;
4  }
```

#### CODE 13. Making an image responsive with CSS3

Figure 14 shows that the `max-width` CSS3 property could be applied to the most of the fixed-width media, such as videos or Adobe Flash objects.

```
1  img, embed, object, video {max-width: 100 %;}
```

#### CODE 14. Making media responsive with CSS3

There are a lot of image file formats. Which ones should be used in responsive web design? The PNG format is the best solution because it supports lossless data compression and transparency. That is why, PNG is useful in small graphics especially for website decoration purposes. JPEG is the best solution in the context of large images and photographs. JPEG is a lossy compressed file format. Other graphics formats have too strong disadvantages. That is why, JPEG and PNG are the choice of a responsive web design.

Will an image have a fine detalization and impressive effects if it is stretched on the screen for more then its own width? The vector format could solve this problem. How could SVG be used in scaling images? I answered this question in Section 3.7.2, but there is another option - to add retina support.

How to make an image responsive? It is possible with the `srcset`, `sizes` and `wrap-per` tag `<picture>`. The tag `<picture>` in conjunction with the new attributes specifies the smart settings. It could change image `src` attribute when necessary. It allows a developer to configure the browser to select the appropriate picture when needed.

To summarize, flexible images should respond to the retina and load different images for different pixel density, respond to the width of the viewport and load different images for different expressions, operate with a new image formats in a “progressive” way and be compressed without quality loss.

### 3.7.2 Scalable Vector Graphics (SVG)

Scalable Vector Graphics is graphics described by mathematical functions. Size is not important. SVG is infinitely scalable in any direction without the loss of quality.

- SVG scales automatically and has no quality lost.
- Dimensions are specified in regular CSS pixels.
- SVG is supported by IE9 and all modern browsers.
- It has a relatively small file size and could be compressed with SVGZ.

The main problem when using SVG is a weak old browsers support and the complexity of creating a delay options that reflect common bitmaps, when the vector is not supported. Nowadays the situation got changed for the better. The problem only occurs with IE8 and the browsers for Android 2.3 and earlier.

The SVG file could be added to the HTML page with the `<img>`, `<object>` or `<svg>` tag like it is shown in Code 15.

```

1  
2
3  <object data="file.svg" type="image/svg+xml" ... />
4
5  <svg width="100" height="100">
6      <circle cx="50" cy="50" r="40" stroke="green"
7      stroke-width="4" fill="yellow" />
8  </svg>

```

## CODE 15. Three methods of adding SVG to the web page

### 3.7.3 Icon fonts

Icon fonts are fonts where each character is represented by an icon. Before the advent of icon fonts designers and front-end developers used to make web decorations with raster icons, later with sprites. It was beautiful but quite uncomfortable. Today's web design changed and vectors replaced raster graphics. The operation principle of icon fonts is simple. All we need is to choose the icons needed and connect them to the web page. Figure 16 shows Fontello icon font website page, where the needed icons could be choosed. CSS indicate such icon properties as the size, the color, the shade, etc.

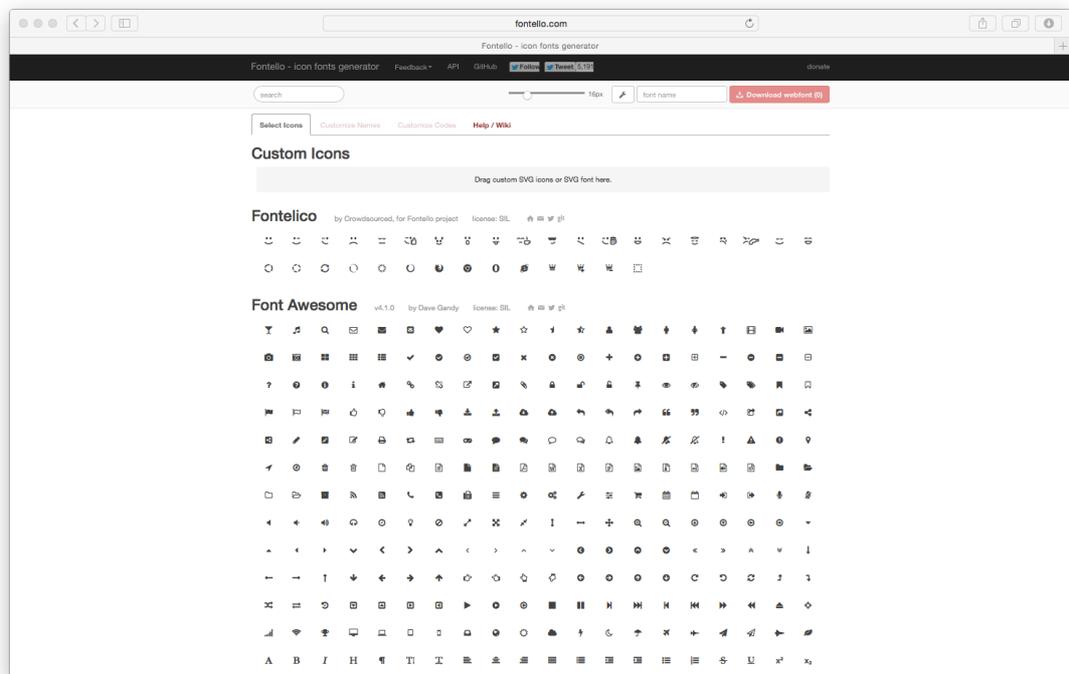


FIGURE 16. Some of the Fontello font icons

Icon fonts have the following advantages:

- There are plenty of ready-made icon sets with the overall style.
- The authors of icon fonts sets offer handy icon search.
- Icon fonts are equally clearly displayed on all screens, even retina.

- The front-end developer workflow got simplified in case of the layout development.
- Icons are loaded in a single font file and it became easier and more profitable to use a CDN.
- We can choose an icon color and size.
- There is the ability to manipulate the icon background, for example, to set the fill color or image.
- We can change the icons styles when hovering on the icon to add more interactivity.
- Modern CSS3 styles, such as shadows, transformation and animation could be applied to the icon.

Icon fonts have the following disadvantages:

- It is difficult to change the icons appearance and more vector graphics and relevant software packages knowledge is needed.
- The entire set of icons is loaded, even if only a few icons are used on the page. This problem could be solved by using font icons generators, like Fontello.
- After changing some icon a new set of icons should be built and converted to different file formats. It is a bit challenging without skills and appropriate software. This problem could partially be solved with the help of font icons generators, like Fontello.

Comparing the pros and cons of icon fonts it is absolutely clear that disadvantages have less weight than the advantages. Only one major shortcoming can be identified – some knowledge is needed to work with the icon fonts.

#### **3.7.4 SVG vs Icon Fonts**

Because the preservation of the quality of resizing is a priority in responsive web design, both SVG and icon fonts certainly fit. The main difference is that we can not apply styles exactly to some part of the icon in an icon font. On the other hand, SVG provides

such an opportunity. That makes the choice between SVG and icon fonts up to the developers' skills and the project needs.

### 3.7.5 Flexible video

Embedding video files to the web page is a bit more complicated than it could firstly appear. Embedding an HTML5 video is an easy task. The only thing needed is to apply `max-width` technique like with the responsive images. Code 16 shows how the `max-width` property should be used with video selector.

```
1  video {
2      max-width: 100%;
3      height: auto;
4  }
```

CODE 16. Making HTML5 responsive video with CSS

Things become a bit more complex when third party video files should be embedded using `iFrame`. The best solution is to add a relative wrapper block and to fully stretch `iFrame` by the wrapper dimensions like it is shown in Code 17.

```
1  .video-wrapper {
2      position: relative;
3      padding-bottom: 56.25%; /* 16:9 */
4      padding-top: 25px;
5      height: 0;
6  }
7  .video-wrapper iframe{
8      position: absolute;
9      top: 0;
10     left: 0;
11     width: 100%;
12     height: 100%;
13 }
```

CODE 17. Making `iFrame` responsive video with CSS

In this example `padding-bottom` is needed to maintain the 19:6 ratio. 56.25% padding was obtained after dividing 9 by 16. After these manipulations the videos become responsive.

### 3.7.6 Flexible audio

There is nothing complex in making audio flexible. The controls are hidden by default, but could be shown by adding the controls' attribute like shown in Code 18:

```
1 <audio controls src="some_audio.ogg">
2   <p>Your browser doesn't support the HTML audio
3   tag. Be sad.</p>
4 </audio>
```

CODE 18. HTML5 audio with controls

Only the `width: 100%` property should be added to the audio selector in order to make this audio flexible. Code 19 shows how to make a responsive audio.

```
1 audio {width: 100%;}
```

CODE 19. Making the audio flexible with CSS

## 3.8 Special cases

The responsive approach has some special cases that should be considered while developing to get the best quality result in case of UX and UI. These special cases include touch optimization, responsive mobile navigation and old browsers' compatibility. They are not a mandatory part of the responsive design approach.

### 3.8.1 Touch optimization

Touch optimization is a technique aimed to increase the mobile user's usability. Sometimes it is really hard to catch the link or tap the desired element while using mobile device for surfing through the network. There is only one solution recommended by Apple and Google companies – increasing the element's internal margins so that the active area become at least 44px in height. This could be done with padding property and media queries.

Let's assume that the text size in the block is 16px and a line height is 24px. This means another 20px are needed to fit the requirement. By dividing this value by two (top and bottom) 10px padding could be found out. The responsive web design formula (Formula

1) is required to find out what the padding in “em” units will be. Therefore, in order to make all the links in the element with the class “.block” the following style properties should be added from Code 20.

```

1  .block a {
2      display: block;
3      padding: 0.625em 0; /* 10px / 16px = 0.625 */
4  }

```

CODE 20. Defining touch optimized links

### 3.8.2 Responsive mobile navigation

Mobile navigation is a very delicate point in responsive web design. It is definitely clear that it should not be the same as on the normal screen. Let us consider the example of the responsive navigation in case of my personal website. Figure 17 shows how the navigation menu looks like in the normal FullHD screen.

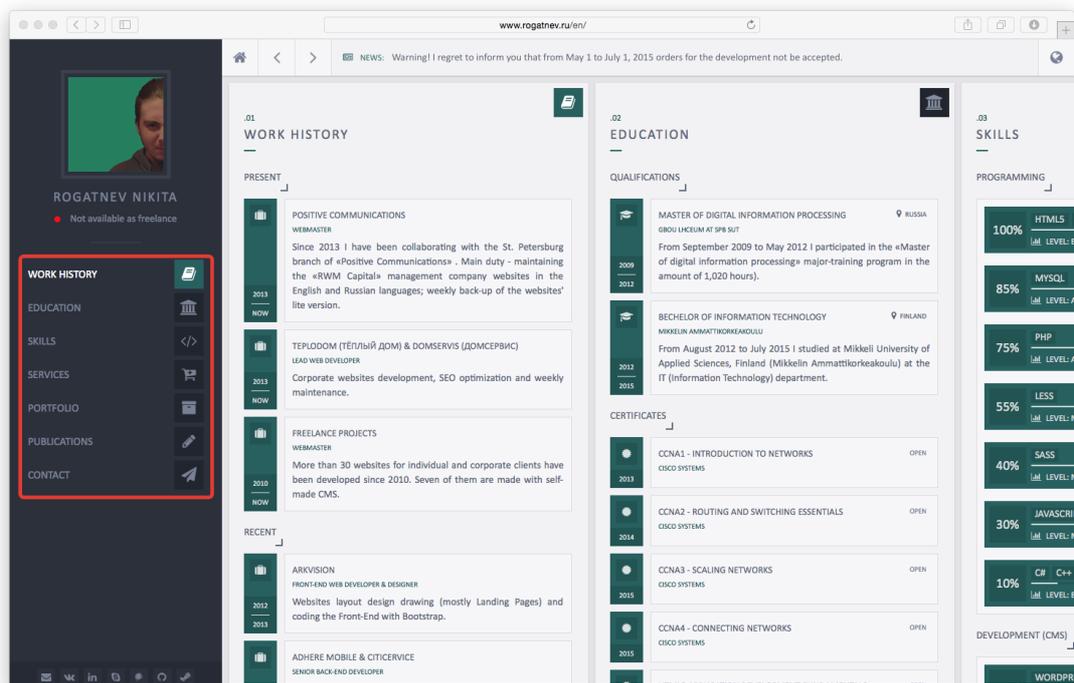


FIGURE 17. Menu position on a normal screen example

Figure 18 shows how the navigation menu looks like on the mobile device screen. This is only one possible option for implementing responsive navigation. The main point here is to find out the balance between the navigation position, width and decoration in

order to minimize content overlay. In case the menu is the same on both variations, on the mobile device it will rather occupy too much canvas, so that a user has to scroll to the content, or it will have too small menu items making it hard to tap on the required one.

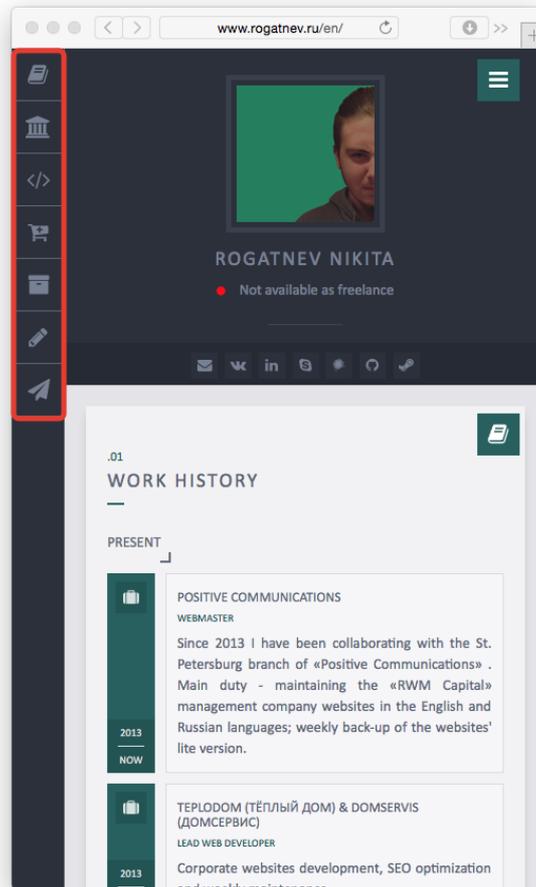


FIGURE 18. Menu appearance on a mobile device example

To summarize, responsive mobile navigation should meet the following criteria:

- Navigation should not take up a lot of precious canvas space.
- Navigation should be simple and clean in order not to make a user confused.
- Navigation does not have to be the same on a wide variety of devices, but all the navigation objects should have the same order.
- If there is a huge set of nested menu items, the use of drop down lists or a toggle button will be the best solution.

### 3.8.3 Old browsers' compatibility

The operating principle of responsive web design is based on CSS3 media queries which are not supported, for example, by Internet Explorer 8. Despite this fact, IE8 recently celebrated its fifth anniversary and definitely seems to be a part of a history. But according to the W3C statistics, IE8 is still more popular than IE9 and IE10. The good news is that there is an open CSS library `css3-mediaqueries.js` which adds the support of media queries in the older versions of IE, Firefox and Safari. Such a high percentage of IE8 on the market is due to the fact that it is the latest version of Internet Explorer that could be run under Windows XP.

Microsoft has stopped the official support for this operating system in April 2014. Accordingly, large corporations and government agencies will have to upgrade to the new version of the OS. Up-to-date W3C 2015 statistics show the decreasing vector of its market share, and therefore it seems that IE8 will disappear soon. Is it worth investing money and power in the past today? My opinion is obviously no.

## 4 RESPONSIVE WEB DESIGN IMPLEMENTATION

In this part of my study I implemented a one-page website with a responsive web design. First of all, I stated the project requirements and prepared the workspace: created project folders, files and uploaded them to the web server. The next step was to draw and design the front-end in Adobe Photoshop CS6. After that the development process started. Development process consists of five main parts: design, the layout building, CSS definition, media query and viewport definition. The development model was classical waterfall.

### 4.1 Requirements analysis

The idea of a project work is to implement a one-page responsive website without using frameworks or grid generators – just clean HTML5 and CSS3. Responsiveness must not be done with the help of JavaScript or other scripting language. The website should match the responsive design principles described in the theoretical part of the study. In order to demonstrate meeting to the responsive design requirements the website will be filled with different media content and tested on different devices. The screenshots of the results are introduced in Section 4.8.

### 4.2 Workspace preparation

The responsive website project is stored on the Apache HTTP Server. The project's absolute folder path is <http://www.rogatnev.ru/works/mamk/responsive/>. I used the Panic Transmit 4.4.7 FTP connection to the web server to create folders and files on the web server. I created four folders on the server: "css" for Cascading Style Sheets, "img" for images or some other graphics and "media" for audio and video. Also styles.css and index.html blank files were created. After that, I uploaded the audio file and picture which I did add to the web site. Now the project root directory "responsive" looks like it is shown in Figure 19.

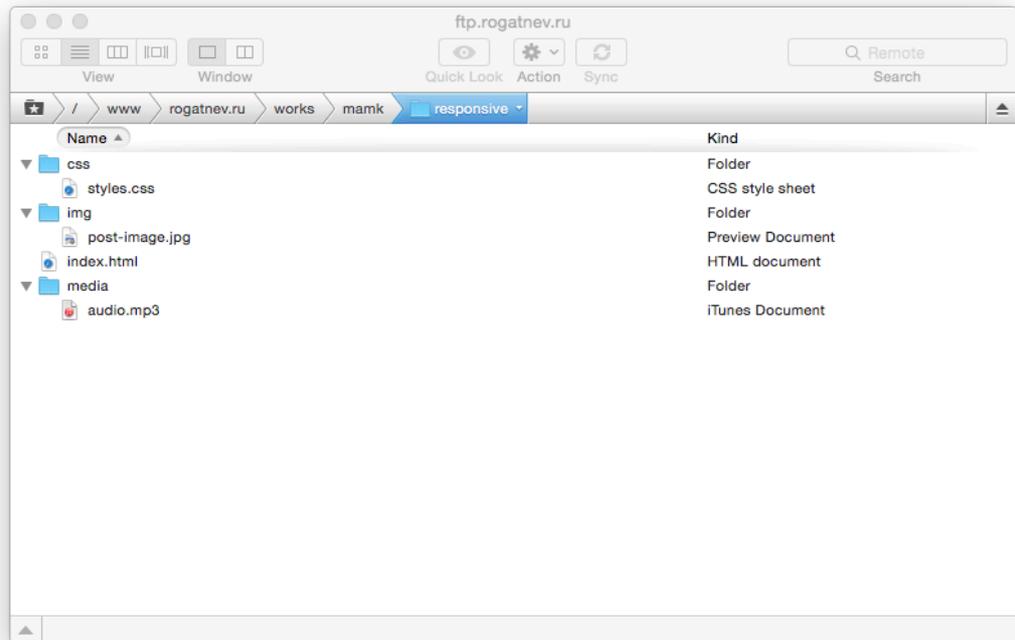


FIGURE 19. Project's folders and files

### 4.3 Project design

Before starting the project implementation a design sketch (Figure 20) should be drawn to get an idea of how the final result should look like. Figure 21 shows the design layout of the project made in Photoshop CS6.

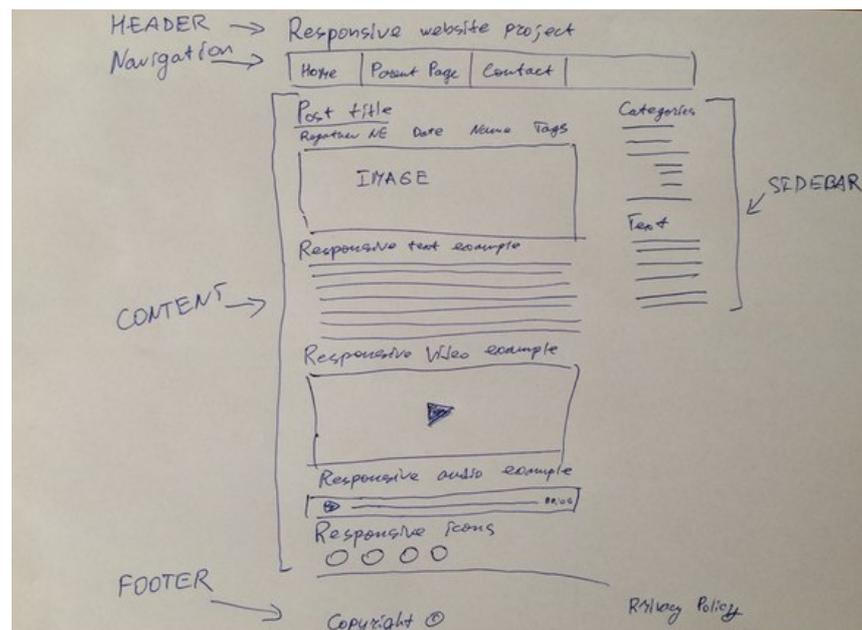


FIGURE 20. Project's design sketch



FIGURE 21. Project's design

#### 4.4 The layout building

The first step is to make HTML5 page blank canvas in index.html file. Code 21 consists of three main HTML parts: the <html> wrap, <head> and <body> tags. The

<head> container consists of meta tags which are used to store information intended for browsers and search engines. The <body> container is designed to store the content of the web page to be displayed in a browser window.

```

1  <!DOCTYPE HTML>
2  <html>
3  <head>
4      <meta charset="UTF-8" />
5      <title>Responsive website project</title>
6  </head>
7  <body>
8  </body>
9  </html>

```

CODE 21. Blank HTML5 canvas code in index.html file

After that, the header, navigation, content, sidebar and footer tags should be added in the <body> tag to define the web page's containers.

There is a headline of the project – Responsive website project in the header block (Appendix 1, code lines 12-18). The navigation block consists of three menu items and three child items (Figure 22) (Appendix 1, code lines 20-34). The content container is the biggest one. It contains post title, post author, date, category, tags, article and responsive media (Appendix 1, code lines 36-78). There is also a sidebar with the categories menu and the text block (Appendix 1, code lines 80-105). The last container is a footer (Appendix 1, code lines 107-116). It consists of a privacy policy link and developer's name. The container <div id="wrapper"> will be useful for adjustment control when using responsive media queries.



FIGURE 22. Navigation structure

Code 22 shows the contents of the <body> tag. Lines 3, 5, 7, 9, 11, 13, 15, 17, 19 and 21 are the code comments and they are not displayed to users in a browser's window. Comment lines and indents are very useful in case of quick code navigation.

```
1 <div id="wrapper">
2   <!-- HEADER -->
3   <header></header>
4   <!-- HEADER END -->
5
6   <!-- NAVIGATION -->
7   <nav></nav>
8   <!-- NAVIGATION END -->
9
10  <!-- CONTENT -->
11  <section class="content"></section>
12  <!-- CONTENT END -->
13
14  <!-- SIDEBAR -->
15  <aside class="sidebar"></aside>
16  <!-- SIDEBAR END -->
17
18  <!-- FOOTER -->
19  <footer></footer>
20  <!-- FOOTER END -->
21 </div>
```

CODE 22. Main page containers markup in index.html file

Semantically, a header is used to demarcate any content that is summarily important to a page or section of a page. It can be used to encapsulate headings or heading groups (contained in the new hgroup element), relevant navigational aids, and introductory content. As such, it makes a perfect container for the title of our page and the list of anchors to each article within the page. (Gustafson, 2011)

The next step is to fill in this container with other HTML5 tags and text content. All the source code of the index.html file could be found in Appendix 1. Now the project page looks like shown in Figure 23.

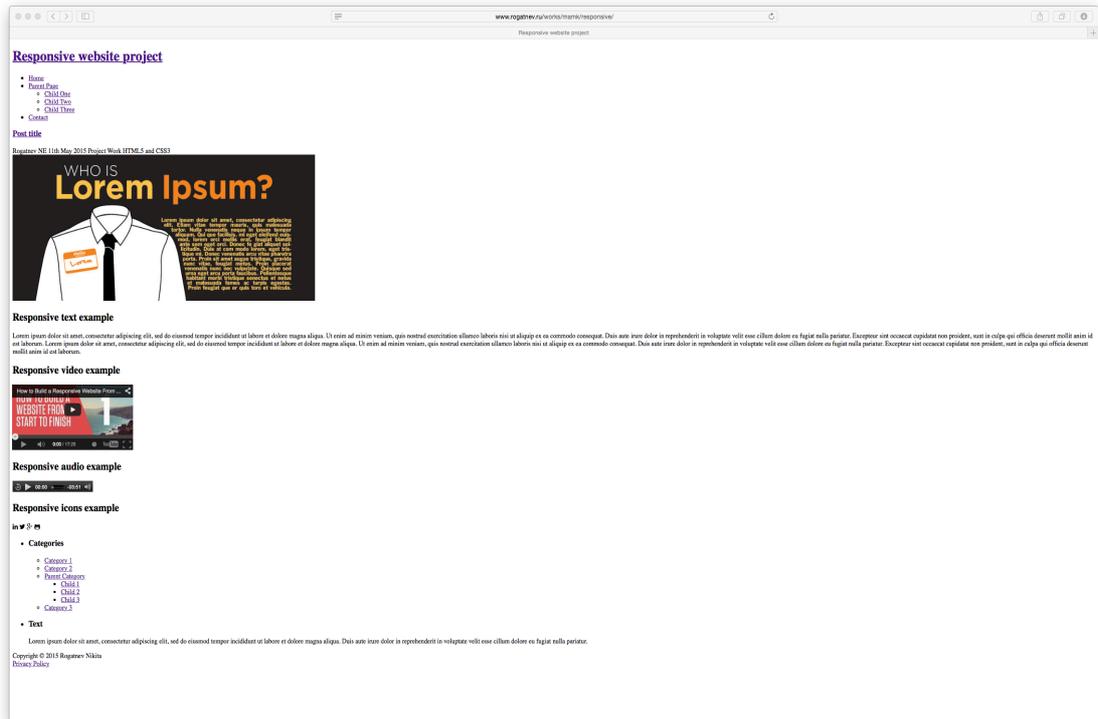


FIGURE 23. Project's index.html file without styling

Now it is time to add some responsive icons to the web page. I used BootstrapCDN by MaxCDN to connect the Font Awesome icons set with the following line inside the <head> tag (Code 23):

```
1 <link href="//maxcdn.bootstrapcdn.com/font-awesome/4.3.0/css/font-awesome.min.css"
  rel="stylesheet" type="text/css">
```

CODE 23. Font Awesome icons set in index.html file

In order to insert LinkedIn, Twitter, Google+ and GitHub icons the following code should be added (Figure 24). Also a font-family: 'FontAwesome'; declaration should be added to the .fontawesome-icons selector styles.

```
<div class="fontawesome-icons">
  <i class="fa fa-linkedin"></i>
  <i class="fa fa-twitter"></i>
  <i class="fa fa-google-plus"></i>
  <i class="fa fa-github-alt"></i>
</div>
```

FIGURE 24. Responsive social icons HTML code

The last thing to do is to connect a CSS file `styles.css` to the `index.html` file. In case the style sheet is located in a separate file it should be included by the use of the `<link>` tag. The `href` attribute indicates the link to the file. Code 24 should be inserted into the `<head>` tag of the page.

```
1 <link href="./css/styles.css" rel="stylesheet"
  type="text/css">
```

CODE 24. Connecting CSS file into the `index.html` file

#### 4.5 CSS definition

The second step of the RWD project development is writing the elements' styles to make the website have layout and decoration elements. There are six main parts in the project CSS file. First of all, the general layout declarations (Code 25) should be stated. They include text line height, font family and size, links' color and margins:

```
1  html {
2      line-height: 1.6em;
3  }
4  body {
5      font-family: arial, sans-serif;
6      font-size: 100%;
7      padding: 0;
8      margin: 0;
9      padding: 0 2%;
10 }
11 #wrapper {
12     font-size: 0.8em;
13     max-width: 960px;
14     margin: 0 auto;
15 }
16 .sidebar a, article a, header a, footer a {
17     color: #CC3300;
18 }
```

CODE 25. General styles in `styles.css` file

The next step is to define styles for the header block. Header declarations include block dimensions, margins and position (Code 26). In the project design the main heading has no underlining. That is why line 8 turns it off.

```

1  header {
2      padding: 1em 0;
3      margin: 0px;
4      float: left;
5      width: 100%;
6  }
7  header a {
8      text-decoration: none;
9  }

```

CODE 26. Header styles in styles.css file

Code 27 consists of declarations for the navigation menu and its elements. Menu consists of three buttons: Home, Parent Page and Contact. I used the `<ul>` tag together with the `<li>` tag to create unordered lists. The Parent Page menu item has a dropdown submenu with three more items (Figure 22). That is why there is such multiple nesting in CSS and HTML. These styles define how the navigation and menu items should look like: their color, size, background, margins, borders etc.

```

1  nav {
2      display: block;
3      margin: 0 0 2em;
4      padding: 0px;
5      float: left;
6      width: 100%;
7      background-color: #181919;
8  }
9  nav ul ul {
10     display: none;
11 }
12 nav ul li:hover > ul {
13     display: block;
14 }
15 nav ul {
16     padding: 0;
17     list-style: none;
18     position: relative;
19     display: inline-table;
20     z-index: 9999;
21     margin: 0px;
22     float: left;
23     width: 100%;
24 }
25 nav ul:after {
26     content: "";
27     clear: both;
28     display: block;
29 }

```

```
30 nav ul li {
31     float: left;
32 }
33 nav ul li:hover a {
34     color: #fff;
35 }
36 nav ul li a {
37     display: block;
38     padding: 1em;
39     font-size: 1.125em;
40     color: #ccc;
41     text-decoration: none;
42     margin: 0px;
43     background-color: #000;
44     border-right: 1px solid #333;
45 }
46 nav ul li:last-of-type a {
47     border-right: 1px solid transparent !important;
48 }
49 nav ul ul {
50     background: #5f6975;
51     border-radius: 0px;
52     padding: 0;
53     position: absolute;
54     top: 100%;
55     width: auto;
56     float: none;
57 }
58 nav ul li:hover {
59     background: #5f6975;
60     color: #FFF;
61 }
62 nav ul ul li a:hover {
63     background-color: #4b545f;
64 }
65 nav ul ul li {
66     float: none;
67     border-bottom: 1px solid #444240;
68     position: relative;
69 }
70 nav ul ul li a {
71     padding: 0.5em 1em;
72     font-size: 1em;
73     width: 10em;
74     color: #fff;
75 }
76 nav ul ul ul {
77     position: absolute; left: 100%; top:0;
78 }
```

CODE 27. Navigation styles in styles.css file

This step is very important because it includes responsive styles for media. Lines 37-48 in Code 28 make the embedded YouTube video responsive, 49-50 lines are intended to make the audio responsive. `max-width:100%` and `height:auto` properties make all images in the content block responsive.

```
1  section.content {
2      width: 70%;
3      float: left;
4  }
5  .content article {
6      width: 100%;
7      float: left;
8      padding: 0 0 1em;
9      margin: 0 0 1em;
10     border-bottom: 1px solid #ddd;
11     text-align: justify;
12 }
13 article .entry {
14     clear: both;
15     padding: 0 0 1em;
16 }
17 article img {
18     max-width: 100%;
19     height: auto;
20 }
21 h1.post-title {
22     font-size: 1.8em;
23     margin: 0;
24     padding: 0;
25 }
26 .entry.post-meta {
27     color: #888;
28 }
29 .entry.post-meta span {
30     padding: 0 1em 0 0;
31 }
32 .entry.post-content {
33     font-size: 1.125em;
34     margin: 0;
35     padding: 0;
36 }
37 .video-wrapper {
38     position: relative;
39     padding-bottom: 56.25%; /* 16:9 */
40     height: 0;
41 }
42 .video-wrapper iframe{
43     position: absolute;
44     top: 0;
45     left: 0;
46     width: 100%;
```

```

47     height: 100%;
48     border:0;
49 }
50 audio{
51     width: 100%;
52 }
53 .fontawesome-icons i{
54     font-size: 2em;
55     color: black;
56     margin-right: 1em;
57     padding: 0;
58 }

```

#### CODE 28. Content block styles in styles.css file

Also, the navigation block sidebar block has `<ul>` tags together with the `<li>` tags in order to create unordered nested lists. The first selector's declaration in Code 29 determines the width of this block. All the other declarations are about lists and their elements styles and position.

```

1  aside.sidebar {
2      width: 25%;
3      float: right;
4  }
5  aside.sidebar ul {
6      width: 100%;
7      margin: 0px;
8      padding: 0px;
9      float: left;
10     list-style: none;
11 }
12 aside.sidebar ul li {
13     width: 100%;
14     margin: 0px 0px 2em;
15     padding: 0px;
16     float: left;
17     list-style: none;
18 }
19 aside.sidebar ul li ul li {
20     margin: 0px 0px 0.2em;
21     padding: 0px;
22 }
23 aside.sidebar ul li ul li ul li {
24     margin: 0px;
25     padding: 0px 0px 0px 1em;
26     width: 90%;
27     font-size: 0.9em;
28 }
29 aside.sidebar ul li h3.widget-title {

```

```
30     width: 100%;
31     margin: 0px;
32     padding: 0px;
33     float: left;
34     font-size: 1.45em;
35 }
36 aside.sidebar ul li .textwidget{
37     text-align: justify;
38 }
```

CODE 29. Sidebar styles in styles.css file

There is one more block left without styling. The footer consists of two parts: author name in the left side and privacy policy link on the right side. Such positioning could be easily done with `float` property (Code 30).

```
1  footer {
2      width: 98%;
3      float: left;
4      padding: 1%;
5      background-color: white;
6      margin-top: 2em;
7  }
8  footer .footer-left {
9      width: 45%;
10     float: left;
11     text-align: left;
12 }
13 footer .footer-right {
14     width: 45%;
15     float: right;
16     text-align: right;
17 }
```

CODE 30. Footer styles in styles.css file

During this step styles markup is finished. The result can be seen in Figure 25. The entire styles.css stylesheet file can be found in Appendix 2.

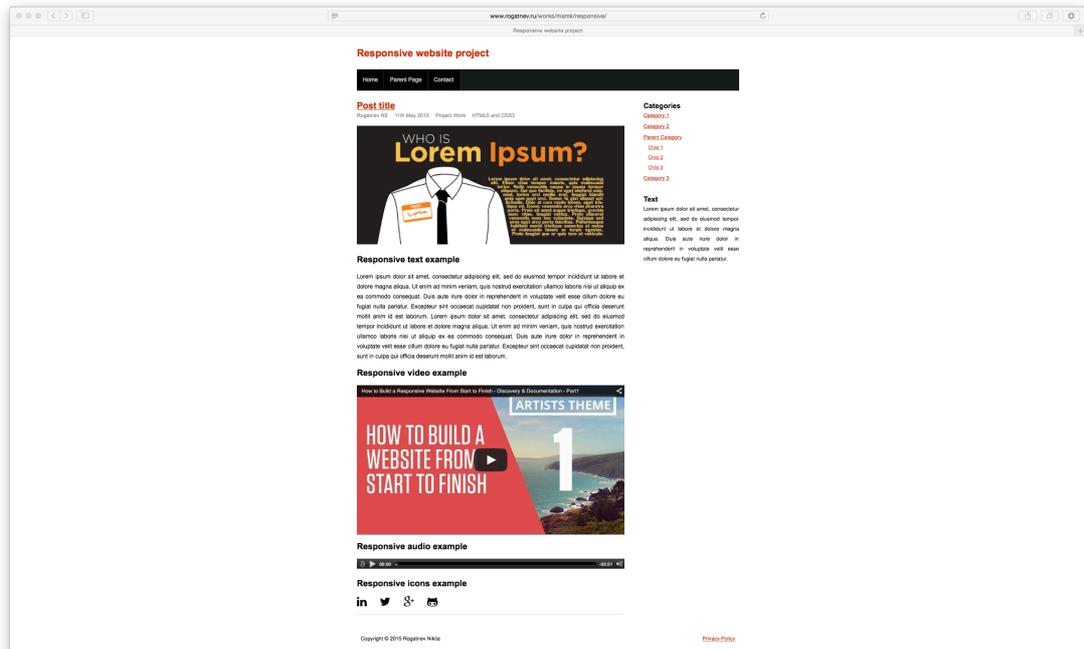


FIGURE 25. Styled project HTML page

## 4.6 Media query and viewport definition

Media query and viewport are the last two things that should be implemented to finish the project work. Media query should be inserted at the end of the styles.css file. There are only two columns in the project and, therefore, one query is enough. It activates when a device screen width is less than 770 pixels (Code 31).

```

1  @media screen and (max-width:770px){
2      #wrapper {
3          font-size: 0.6875em;
4      }
5      section.content, aside.sidebar {
6          width:100%;
7      }
8  }
```

CODE 31. Media query defining in styles.css file

Finally, the viewport meta tag should be defined by inserting the following line inside the <head> tag (Code 32):

```

1  <meta
    name="viewport"
    content="width=device-width,
```

```
initial-scale=1.0,  
minimum-scale=1.0,  
maximum-scale=1.0,  
user-scalable=no" >
```

CODE 32. Viewport defining in index.html file

Such a meta viewport tag tells the browser that:

- When the page loads it must not be scaled.
- The page must take the entire width of the mobile browser.
- Any custom scaling is prohibited.
- Only horizontal and vertical scrolling are available.

#### 4.7 Result screenshots

Now the responsive website project is ready. It looks nice and works nice on all devices from smartphones to TVs. Figures 26-30 show how the project looks like on most of the Apple devices. Here is the list of screenshots available: iPad 768x1024 resolution in the portrait mode (Figures 26 and 27), Macbook with 1280x800 resolution (Figure 28) and iPhone with 480x320 resolution in the landscape mode (Figure 29).

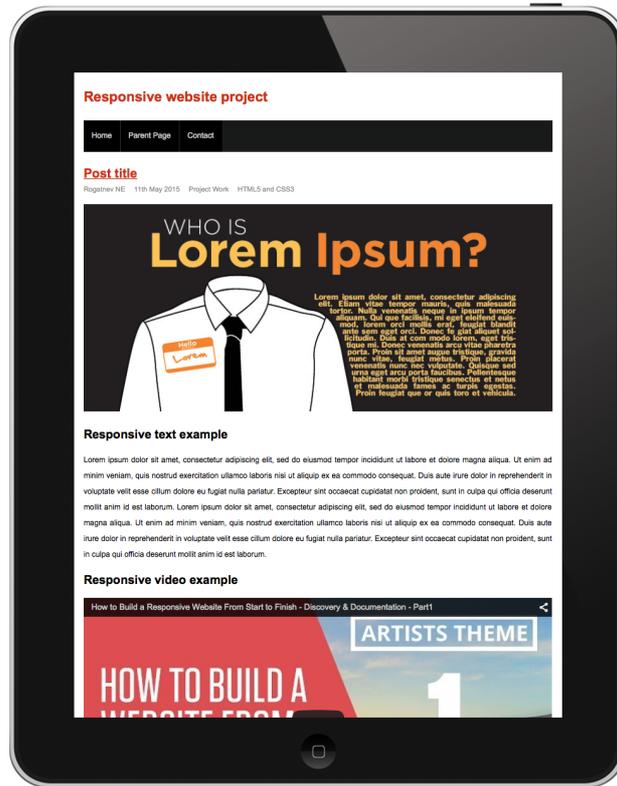


FIGURE 26. iPad portrait 768x1024

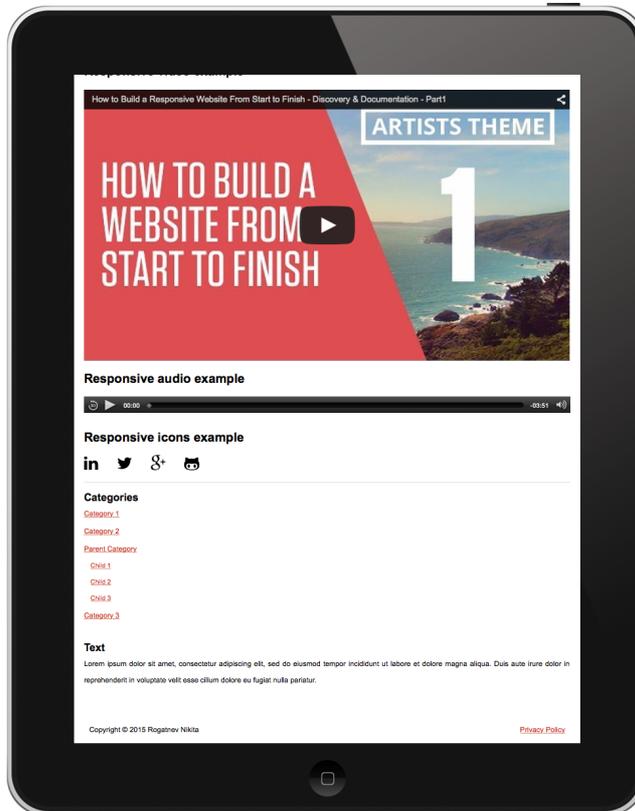


FIGURE 27. iPad portrait 768x1024

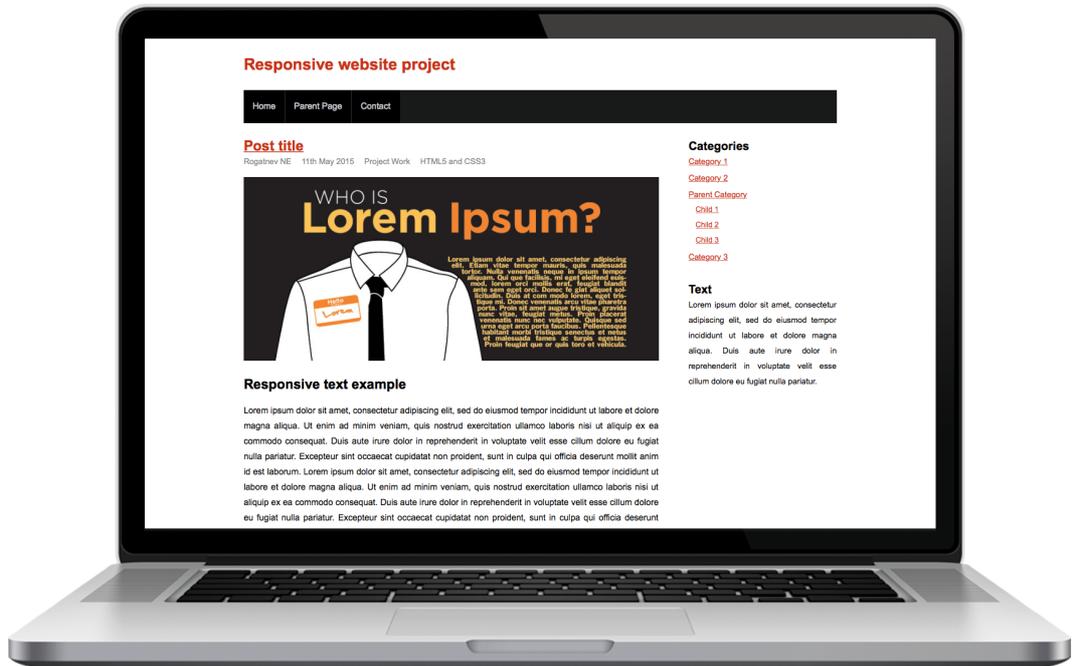


FIGURE 28. Macbook 1280x800

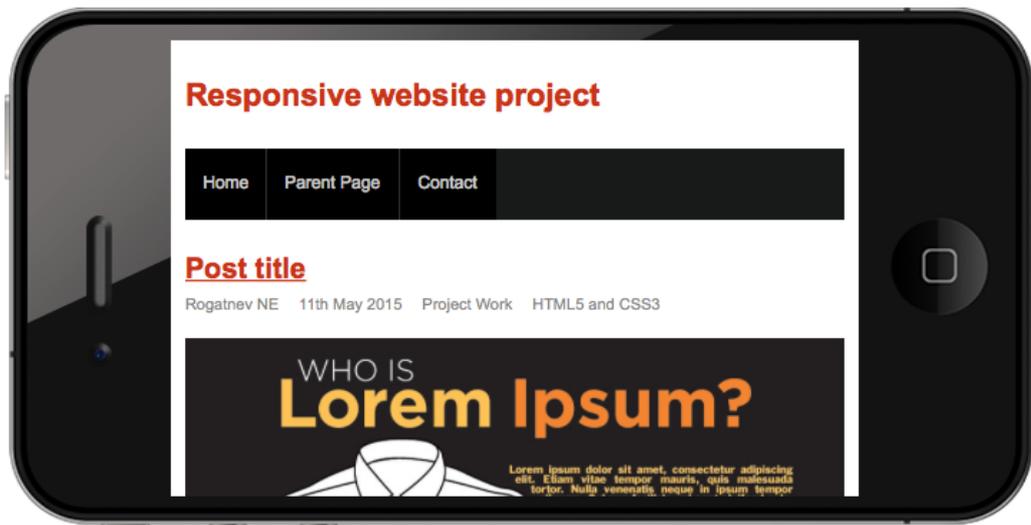


FIGURE 29. iPhone landscape 480x320

## 5 CONCLUSIONS

In this thesis I described everything regarding to the responsive web design. First of all, I summed up and compared all the possible website layout approaches. The disadvantages of fixed, fluid, adaptive and hybrid layouts are so significant that it is undoubtedly clear that responsive approach is the best one.

After that, in the theoretical part I defined what the responsive design was and described its main principles and techniques. To summarize, responsive design consists of the following: responsive layout with media queries to fit all possible devices, viewport and responsive typography and media.

In the practical part I implemented one page responsive website without using frameworks or grid generators – just clean HTML5 and CSS3. In order to improve the complexity of the tasks multiple responsive media was included into the project. The project website includes responsive iFrame with a video from YouTube, responsive HTML5 audio player with music composition stored on the webserver and responsive icons provided by Font Awesome icon font. Typography is also adaptive, because it was set in “em” units, like it was stated in theoretical part of the study. Figure 30 confirms that the project website is really responsive and that all the responsive design principles were used in it.



FIGURE 30. Project website on Apple devices

**BIBLIOGRAPHY**

Cochran, David, 2012. *Twitter Bootstrap Web Development How-To*. Packt Publishing Ltd, UK.

Duckett, Jon 2011. *HTML and CSS: Design and Build Websites*. John Wiley & Sons, Inc., USA.

Gasston, Peter 2013. *The Modern Web: Multi-Device Web Development with HTML5 CSS3, and JavaScript*. No Starch Press, USA.

Gustafson, Aaron 2011. *Adaptive Web Design: Crafting Rich Experiences with Progressive Enhancement*. Easy Readers LLC, USA.

Hay, Stephen 2013. *Responsive Design Workflow*. New Riders, USA.

Kadlec, Tim 2013. *Implementing Responsive Design: Building sites for an anywhere, everywhere web (Voices That Matter)*. New Riders, USA.

LePage, Pete 2014. *Use CSS media queries for responsiveness*. Available in www-format: <https://developers.google.com/web/fundamentals/layouts/rwd-fundamentals/use-media-queries?hl=en/>. Referred [02.05.2015]

Lynch, Patrick 2009. *Web Style Guide: Basic Design Principles for Creating Web Sites*. Yale University Press, UK.

Morton, Lisa. *Responsive Web Design - Why it Matters, Why You Need it*. Available in www-format: <http://www.ddshosting.com/responsive-web-design-matters-need/>. Referred [02.05.2015]

Pingdom AB 2013. *Internet 2012 in numbers*. Available in www-format: <http://royal.pingdom.com/2013/01/16/internet-2012-in-numbers/>. Referred [02.05.2015]

Robbins, Jennifer 2012. *Learning Web Design: A Beginner's Guide to HTML, CSS, JavaScript, and Web Graphics*. Littlechair Inc., Canada.

Shawn, Henry 2007. *Just Ask: Integrating Accessibility Throughout Design*. Available in www-format: lulu.com. Referred [02.05.2015]

WordPress. *Using Media Queries with Custom CSS*. Available in www-format: <https://en.support.wordpress.com/custom-design/custom-css-media-queries/>. Referred [02.05.2015]



```

47         11th May 2015
48     </span>
49     <span class="post-category">
50         Project Work
51     </span>
52     <span class="post-tag">
53         HTML5 and CSS3
54     </span>
55 </div>
56 <div class="entry post-content">
57     
58     <h2>Responsive text example</h2>
59     <p>Lorem ipsum dolor sit amet, consec-
tetur adipiscing elit, sed do eiusmod tempor in-
cididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate
velit esse cillum dolore eu fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id
est laborum. Lorem ipsum dolor sit amet, consecte-
tur adipiscing elit, sed do eiusmod tempor in-
cididunt ut labore et dolore magna aliqua. Ut enim
ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate
velit esse cillum dolore eu fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident,
sunt in culpa qui officia deserunt mollit anim id
est laborum.</p>
60     <h2>Responsive video example</h2>
61     <div class="video-wrapper">
62         <iframe
src="https://www.youtube.com/embed/VcMjo_wczCc" al-
lowfullscreen></iframe>
63     </div>
64     <h2>Responsive audio example</h2>
65     <audio controls src="./media/au-
dio.mp3">
66         <p>Your browser doesn't support the
HTML audio tag. Be sad.</p>
67     </audio>
68     <h2>Responsive icons example</h2>
69     <div class="fontawesome-icons">
70         <i class="fa fa-linkedin"></i>
71         <i class="fa fa-twitter"></i>
72         <i class="fa fa-google-plus"></i>
73         <i class="fa fa-github-alt"></i>
74     </div>
75 </div>
76 </article>

```

```
77     </section>
78     <!-- CONTENT END -->
79
80     <!-- SIDEBAR -->
81     <aside class="sidebar">
82         <ul class="widget-sidebar">
83             <li class="widget widget_categories">
84                 <h3 class="widget-title">Categories</h3>
85                 <ul>
86                     <li><a href="#">Category 1</a></li>
87                     <li><a href="#">Category 2</a></li>
88                     <li><a href="#">Parent Category</a>
89                         <ul class="children">
90                             <li><a href="#">Child 1</a></li>
91                             <li><a href="#">Child 2</a></li>
92                             <li><a href="#">Child 3</a></li>
93                         </ul>
94                     </li>
95                     <li><a href="#">Category 3</a></li>
96                 </ul>
97             </li>
98             <li class="widget widget_text">
99                 <h3 class="widget-title">Text</h3>
100                <div class="textwidget">
101                    Lorem ipsum dolor sit amet, consectetur
adipiscing elit, sed do eiusmod tempor incididunt
ut labore et dolore magna aliqua. Duis aute irure
dolor in reprehenderit in voluptate velit esse cil-
lum dolore eu fugiat nulla pariatur.</div>
102                </li>
103            </ul>
104        </aside>
105        <!-- SIDEBAR END -->
106
107        <!-- FOOTER -->
108        <footer>
109            <div class="footer-left">
110                Copyright &copy; 2015 Rogatnev Nikita
111            </div>
112            <div class="footer-right">
113                <a href="#">Privacy Policy</a>
114            </div>
115        </footer>
116        <!-- FOOTER END -->
117
118    </div>
119 </body>
120 </html>
```

**APPENDIX 2. styles.css file code**

```
1     html {
2         line-height: 1.6em;
3     }
4     body {
5         font-family: arial, sans-serif;
6         font-size: 100%;
7         padding:0;
8         margin:0;
9         padding:0 2%;
10    }
11    #wrapper {
12        font-size: 0.8em;
13        max-width: 960px;
14        margin: 0 auto;
15    }
16    .sidebar a, article a, header a, footer a {
17        color: #CC3300;
18    }
19
20    /* HEADER CSS */
21    header {
22        padding: 1em 0;
23        margin: 0px;
24        float: left;
25        width: 100%;
26    }
27    header a {
28        text-decoration: none;
29    }
30
31    /* NAVIGATION CSS */
32    nav {
33        display: block;
34        margin: 0 0 2em;
35        padding: 0px;
36        float: left;
37        width: 100%;
38        background-color: #181919;
39    }
40    nav ul ul {
41        display: none;
42    }
43    nav ul li:hover > ul {
44        display: block;
45    }
46    nav ul {
47        padding: 0;
48        list-style: none;
49        position: relative;
50        display: inline-table;
```

```
51     z-index: 9999;
52     margin: 0px;
53     float: left;
54     width: 100%;
55 }
56 nav ul:after {
57     content: "";
58     clear: both;
59     display: block;
60 }
61 nav ul li {
62     float: left;
63 }
64 nav ul li:hover a {
65     color: #fff;
66 }
67 nav ul li a {
68     display: block;
69     padding: 1em;
70     font-size: 1.125em;
71     color: #ccc;
72     text-decoration: none;
73     margin: 0px;
74     background-color: #000;
75     border-right: 1px solid #333;
76 }
77 nav ul li:last-of-type a {
78     border-right: 1px solid transparent !important;
79 }
80 nav ul ul {
81     background: #5f6975;
82     border-radius: 0px;
83     padding: 0;
84     position: absolute;
85     top: 100%;
86     width: auto;
87     float: none;
88 }
89 nav ul li:hover {
90     background: #5f6975;
91     color: #FFF;
92 }
93 nav ul ul li a:hover {
94     background-color: #4b545f;
95 }
96 nav ul ul li {
97     float: none;
98     border-bottom: 1px solid #444240;
99     position: relative;
100 }
101 nav ul ul li a {
102     padding: 0.5em 1em;
103     font-size: 1em;
```

```
104     width:10em;
105     color: #fff;
106 }
107 nav ul ul ul {
108     position: absolute; left: 100%; top:0;
109 }
110
111 /* CONTENT CSS */
112 section.content {
113     width: 70%;
114     float:left;
115 }
116 .content article {
117     width:100%;
118     float:left;
119     padding: 0 0 1em;
120     margin: 0 0 1em;
121     border-bottom: 1px solid #ddd;
122     text-align:justify;
123 }
124 article .entry {
125     clear:both;
126     padding: 0 0 1em;
127 }
128 article img {
129     max-width:100%;
130     height:auto;
131 }
132
133 h1.post-title {
134     font-size: 1.8em;
135     margin:0;
136     padding:0;
137 }
138 .entry.post-meta {
139     color: #888;
140 }
141 .entry.post-meta span {
142     padding: 0 1em 0 0;
143 }
144 .entry.post-content {
145     font-size: 1.125em;
146     margin:0;
147     padding:0;
148 }
149 .video-wrapper {
150     position: relative;
151     padding-bottom: 56.25%; /* 16:9 */
152     height: 0;
153 }
154 .video-wrapper iframe{
155     position: absolute;
156     top: 0;
```

```
157     left: 0;
158     width: 100%;
159     height: 100%;
160     border:0;
161 }
162 audio{
163     width:100%;
164 }
165 .fontawesome-icons i{
166     font-size: 2em;
167     color:black;
168     margin-right:1em;
169     padding:0;
170 }
171
172 /* SIDEBAR CSS */
173 aside.sidebar {
174     width: 25%;
175     float:right;
176 }
177 aside.sidebar ul {
178     width:100%;
179     margin: 0px;
180     padding: 0px;
181     float: left;
182     list-style: none;
183 }
184 aside.sidebar ul li {
185     width:100%;
186     margin: 0px 0px 2em;
187     padding: 0px;
188     float: left;
189     list-style: none;
190 }
191 aside.sidebar ul li ul li {
192     margin: 0px 0px 0.2em;
193     padding: 0px;
194 }
195 aside.sidebar ul li ul li ul li {
196     margin: 0px;
197     padding: 0px 0px 0px 1em;
198     width: 90%;
199     font-size: 0.9em;
200 }
201 aside.sidebar ul li h3.widget-title {
202     width:100%;
203     margin: 0px;
204     padding: 0px;
205     float: left;
206     font-size: 1.45em;
207 }
208 aside.sidebar ul li .textwidget{
209     text-align:justify;
```

```
210 }
211
212 /* FOOTER CSS */
213 footer {
214     width: 98%;
215     float:left;
216     padding: 1%;
217     background-color: white;
218     margin-top: 2em;
219 }
220 footer .footer-left {
221     width: 45%;
222     float:left;
223     text-align:left;
224 }
225 footer .footer-right {
226     width: 45%;
227     float:right;
228     text-align:right;
229 }
230
231 /* MEDIA QUERY */
232 @media screen and (max-width:770px){
233     #wrapper {
234         font-size: 0.6875em;
235     }
236     section.content, aside.sidebar {
237         width:100%;
238     }
```