jamk.fi

CREATING RESPONSIVE UI FOR WEB STORE USING CSS

Magdalena Wiciak

Bachelor's Thesis

May 2014

Degree Programme in Information Technology

Technology, communication and transport

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)	Type of publication	Date
WICIAK, Magdalena	Bachelor´s Thesis	14-05-2014
	Pages 48	Language English
		Permission for web publication (X)

Title

CREATING RESPONSIVE UI FOR WEB STORE USING CSS

Degree Programme
Information Technology

Tutor(s)
MIESKOLAINEN, Matti

Assigned by
Descom Oy

Abstract

Nowadays people spend most of their days using mobiles and tablets to access web sites. It is really important to ensure that all customers are able to access a web site no matter where they are and which devices they use. That is why the interest in creating responsive design is increasing.

Responsive Web Design is an approach that provides responding of the web site according to the environment where it is accessed. The thesis shows how to create responsive UI for the web store based on the IBM WebSphere Commerce. The first part the thesis analyses the best practices for creating a responsive UI, which is the usage of flexible grids, images and CSS media Queries.

IBM WebSphere commerce is a software platform for cross-channel commerce. In the third chapter IBM WebSphere Commerce is introduced with Management Center and Commerce Composer. Additionally, the processes of creating a new page and layout are discussed.

The fourth chapter shows how to extend an existing starter store for the client's needs and provide responsive user interface for the new customized web store. The last chapter presents the conclusion concerning this thesis.

The result of the thesis is fully responsive user interface for the Kotipizza web store. In the future the logic of the store behind the UI needs to be added. After all needed tests the web store is to be presented to the customer.

Keywords
Responsive, web design, UI, JSP, RWD, CSS, IBM WebSphere, IBM WebSphere Commerce

Miscellaneous

CONTENTS

ACRONYMS

CSS	Cascading Style Sheets, a style Sheet language used to style the web pages written in HTML
XML	Extensible Markup Language, defines a set of rules for encoding documents
JSP	Java Server Pages, technology designed for creating dynamic WWW pages in following formats: HTML, XHTML, DHTML and XML. JSP uses Java language inside HTML code
MVC	Model-View-Controller architecture, where model represents the business or database code, view – design code and controller – navigational code
RWD	Responsive Web Design, a design that can adjust to the user's screen resolution

1 INTRODUCTION

Nowadays mobile phones and tablets are more and more used for viewing Internet websites, which creates a need to have a web store working for all of the different devices with different screen resolutions. Normally, without using RWD, a web store for mobile phones is created separately from the same store as desktop. What to do when there are different devices with totally different screen resolutions? Should a separate design be created for each of them?

Responsive Web Design is an answer to this question. Responsive Web design gives possibility to dynamically resize and show different content while changing screen resolution. Using RWD gives many benefits. First, it is reduction of code duplication. It also makes conservation much easier. The second benefit is the ability to use the same web store for the devices with a screen resolution that is new on the market. (IBM Info Center 2014)

One of the most important matters to remember while creating responsive UI is that the layout not only needs to look good in every screen resolution but it also must be user-friendly. It is not possible to show all the content from a desktop on devices with a smaller screen, which is why some of the content should be hidden and replaced by other usability.

The thesis shows how to create responsive UI for WebStore. WebStore is created for Kotipizza restaurant. The project is implemented by Descom Oy. There are eight people working with this project, three of them are in UI development team. In the thesis, the default IBM code and the customization made for the purposes of Kotipizza Web Store are analyzed.

2 WEB DESIGN

The following chapter will show the best practices that can be used while implementing the RWD.

2.1 Usage of the percentages

The first thing that needs to be done while creating responsive layout is changing the fixed pixel-based width to percentage width. This one modification can provide that the width of the target element changes correspondingly with the resized page.

Ethan's Marcotte (2011) math formula can be used to be sure that the percentage dimensions are calculated correctly: ***target / context = result***, where the target is the amount of pixels that the target should have and the context is the container to which relation pixels are checked. The result is the searched quantity of pixels expressed by a percentage. (28 – 33)

The example situation for non-responsive websites can be considered. The page width is set to 960 pixels, inside the page is a wrapper that has the width 600px, and inside this wrapper is a column with width 500px. The CSS code could look like this:

.wrapper { width = 600px; }

.column { width = 500px; }

Now if the layout needs to be changed to be responsive; however, leaving the same proportion between widths of each of the element. The percentage values need to be calculated. For wrapper it will be 600 / 960 = 0.625 which means 62.5% and for column: 500/600 = 0.8333333 which equals to 83.333333%. An important issue is that the context for each of these elements is different. The new CSS code looks like this:

.wrapper { width = 62.5%; }

.column { width = to 83.333333%; }

This formula can be used not only to calculate the page width but also other page components such as the size of the margins. The proportion between page width and margins set using percentage will always be the same despite the fact that the size of the browser window will change.

2.2 Fixed and Fluid Layout design

There are a few layout designs that can be chosen while creating a website. Two of them are Fixed and Fluid design. In the fixed layout the whole content is wrapped into container with a fixed size given in pixels, and all components inside have the width measured in pixels or in percentages. On the other hand, in the fluid layout almost all components have dimensions defined in percentages. In fluid layout some of the elements, such as margins can have their widths set in pixels. The whole layout will still adjust to the screen resolution. (Knight 2009)

An example of this in use is shown in Figure 1 and Figure 2.

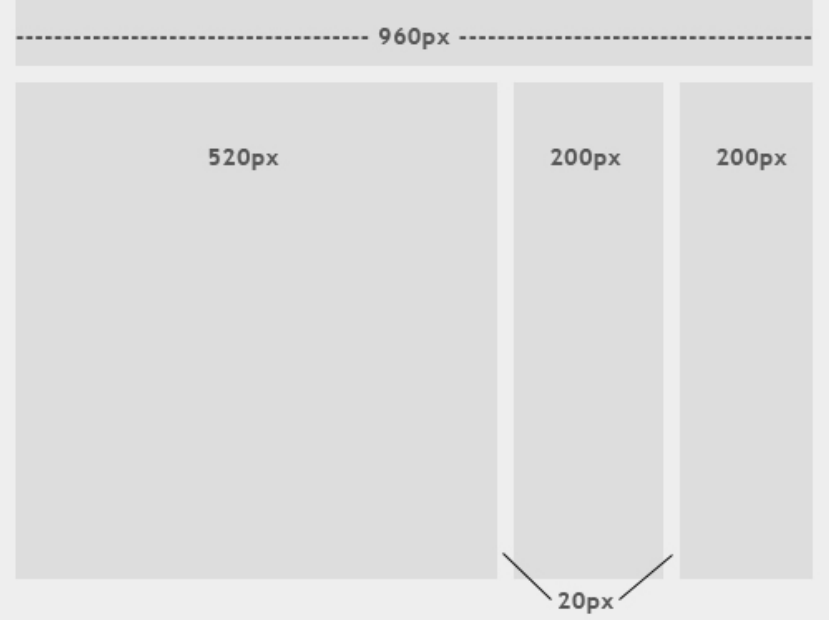

Figure 1. Fixed website design

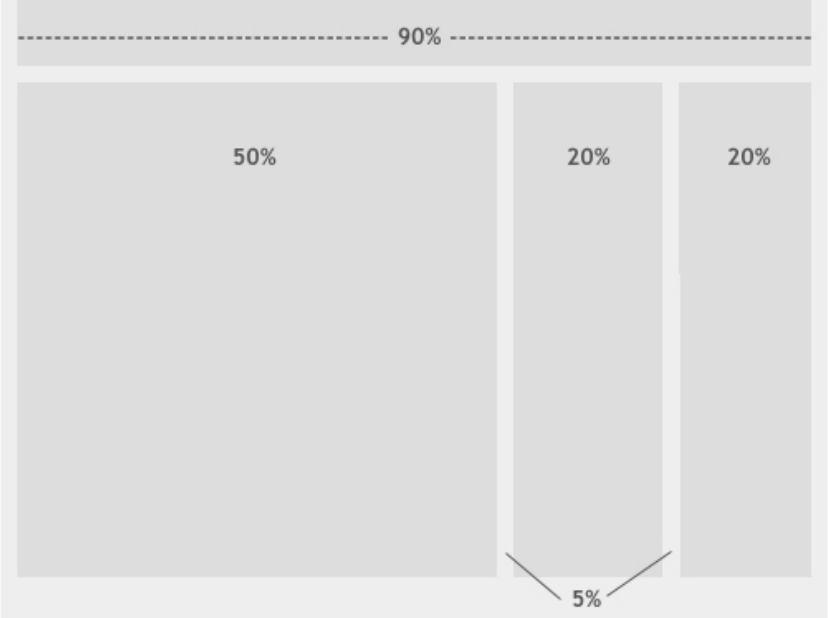

Figure 2. Fluid website design

The major disadvantages of using the fixed layout design are that on the devices with screen resolution smaller than the container not the entire layout is shown. Therefore the horizontal scroll bar is required which decrease the usability. Usage of the fluid layout can eliminate this issue. (Knight 2009)

2.3 Fluid Grid

Fluid Grid is a CSS framework which divides the page layout for rows and columns. The size of the column is fluid which means that the actual size of it is calculated based on the percentage of the context. (Responsive Web Design (RWD) framework 2014)

The 12-column grid is used in the web store. Each of the columns has the width of 8.333333% thus all 12 columns connected together give a total width of 100%.There are no margins between columns in the grid. Columns are defined at CSS file as the ***classes*** col*. The defined col classes can be used to create the Grid system for responsive layout template.

```
/* Columns */
.col1 { width: 8.333333%; }
.col2 { width: 16.666666%; }
.col3 { width: 25%; }
.col4 { width: 33.333333%; }
.col5 { width: 41.666666%; }
.col6 { width: 50%; }
.col7 { width: 58.333333%; }
.col8 { width: 66.666666%; }
.col9 { width: 75%; }
.col10 { width: 83.333333%; }
.col11 { width: 91.666666%; }
.col12 { width: 100%; }
```

The example shows the usage of the col classes in the jsp file.

```
<div class="rowContainer" id="container_${pageDesign.layoutID}">
        <div class="row">
        <div class="col12">
            // Content
        </div>
    </div>
    <div class="row">
        <div class="col12">
            // Content
        </div>
    </div>
    <div class="row">
         <div class="col4 acol12">
            // Content
         </div>
         <div class="col8 acol12" data-slot-id="3">
            // Content
         </div>
    </div>
</div>
```

The page layout of this example is divided into three rows. The first row and the second row have one column with the width of 100%. The last row has two columns: col4 – 33.333333% and col8 – 66.666666%. The width of the columns changes responsively according to the width of the whole page. The page using the example JSP file could look like the page shown in Figure 3.

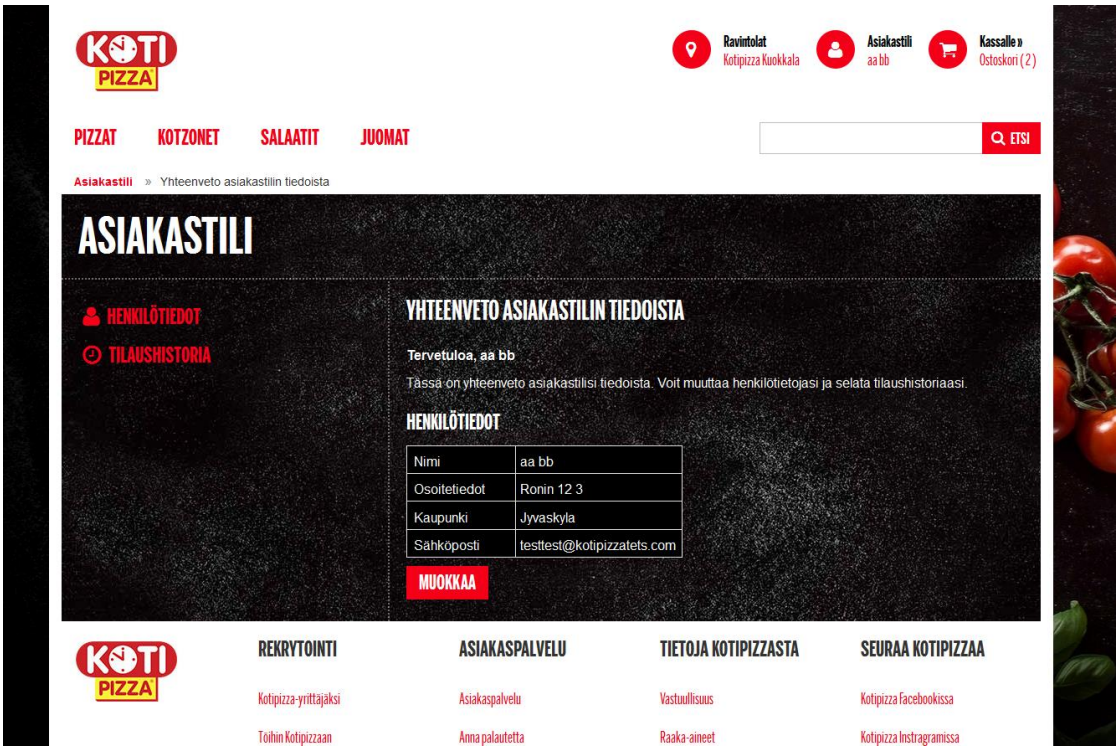

Figure 3. The page based on the example JSP file with the page width 1200px

The fluid grid page layout of the example looks good for a page width bigger than 600px. When the size of the web browser is resized to the smaller resolution the columns start to be too narrow and the layout does not look good and useful anymore (See Figure 4). The layout is broken and the solution for this problem is shown in the following chapter.

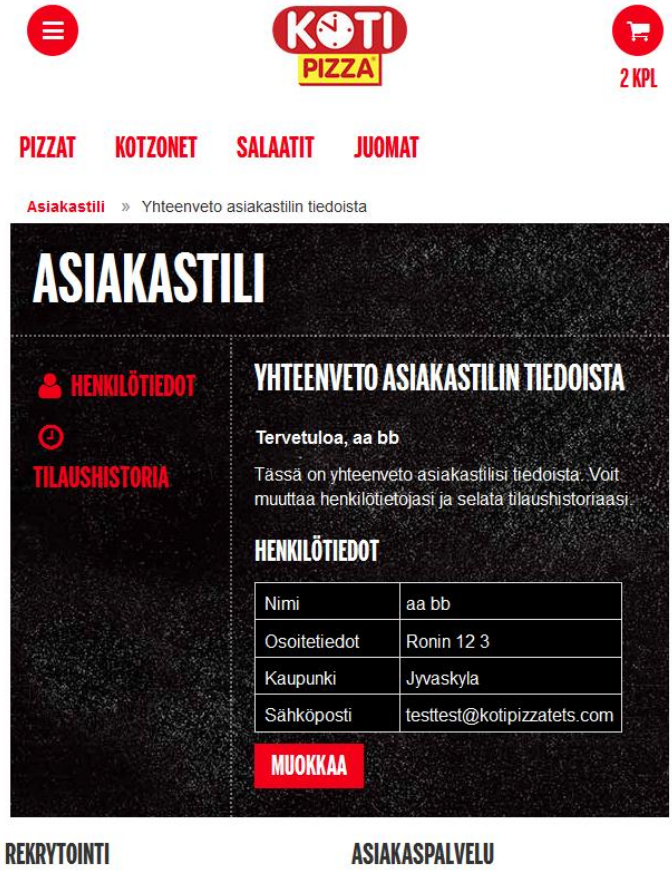

Figure 4. The page based on the example JSP file with the page width 620px. The page layout is broken; the word “tilaushistoria” is no longer on one line with the icon.

2.4 Breakpoints

The breakpoints are the points where the new style sheet should be used because the old layout is broken. The breakpoints are set according to the different screen resolution – page width, when the content of the page is no longer suitable. It does not only mean that the new style needs to be set when the screen size is changing from the bigger to the smaller one, but also when the change is done in the opposite way. (Van Gemert, 2013)

When the breakpoints are defined they can be used in media queries.

2.5 Media Queries

In the previous chapters it was shown how to make the browser resize the content when the screen size is changed; however, as shown in Figure 4 it is not enough.

How to optimize the content that is shown on the page depending on the browser size and device on which it is used? The answer to this is question is the usage of media queries. The media queries allow setting different styles when the screen matches the criteria set in the parameters of the media queries. (Media Queries 2012)

@media screen and (max-width: 600px) { … }

Media queries consist of two parts. The first part is media type. It needs to be set as screen for screen-based devices like Smartphone, tablet or desktop. If a media query is left without media type defined it will apply to all media types.

The second part is the query itself. The query can be split into another two parts: the feature and the value that must be matched. The most interesting features that can be used to create responsive UI are width, height and orientation. The width and the height can be used with ***min*** and ***max*** prefixes. The min width and the max width can apply to screen size and to the browser width. The ***min-device-width*** and ***max-device-width*** check the actual device width, not the screen width. Different queries can be connected together by using the ***and*** keyword.

A media query is the logical expression which means that it can be either true or false. A media query can be true only if the media type of the media queries matches the media type of the devices and all expressions set in the query are true.

@media screen and (max-width: 600px) and (orientation: landscape) { … }

In the example above the browser renders the CSS code only if the viewport width is smaller than 600px and orientation is landscape. If it is not then the part of the code outside a media query is rendered. If the media queries are true only the features set inside media queries are overridden and the rest of the features remain without changes.

```
@media (max-width: 600px) {
    .acol1 { width: 8.333333%; }
    .acol2 { width: 16.666666%; }
    .acol3 { width: 25%; }
    .acol4 { width: 33.333333%; }
    .acol5 { width: 41.666666%; }
    .acol6 { width: 50%; }
    .acol7 { width: 58.333333%; }
    .acol8 { width: 66.666666%; }
    .acol9 { width: 75%; }
    .acol10 { width: 83.333333%; }
    .acol11 { width: 91.666666%; }
    .acol12 { width: 100%; }
    .rowContainer > .row.margin-true {
        padding-left: 0;
        padding-right: 0;
    }
    .row.margin-true > div > div {
        margin-left: 0;
        margin-right: 0;
    }
}
```

2.6 CSS Multiple Classes

The CSS language allows the use of multiple classes to set different styles for the elements. Classes can be added to the class selector as attributes. Each of the class names must be separated by space. (Kyrnin)

```
<div class="col3 acol12"> </div>
```

Usage of the div element from example above and media query defined in last subchapter ensure that if the screen width is wider then 601px, the width from the col3 class is rendered for the div element. Otherwise the screen width fits to the breakpoints set in media query. The browser uses the acol12 class and overrides the width from col3 class.

Using the media queries and the fluid grid it is possible to ensure that the same column on a big screen has only a specified part of the width and on the smaller screen it is positioned with full width.

Figure 5 shows the same page that was shown on Figure 4 but with a new style sheet. Each of the columns used on the page has now width 100% and they are placed one below the other.

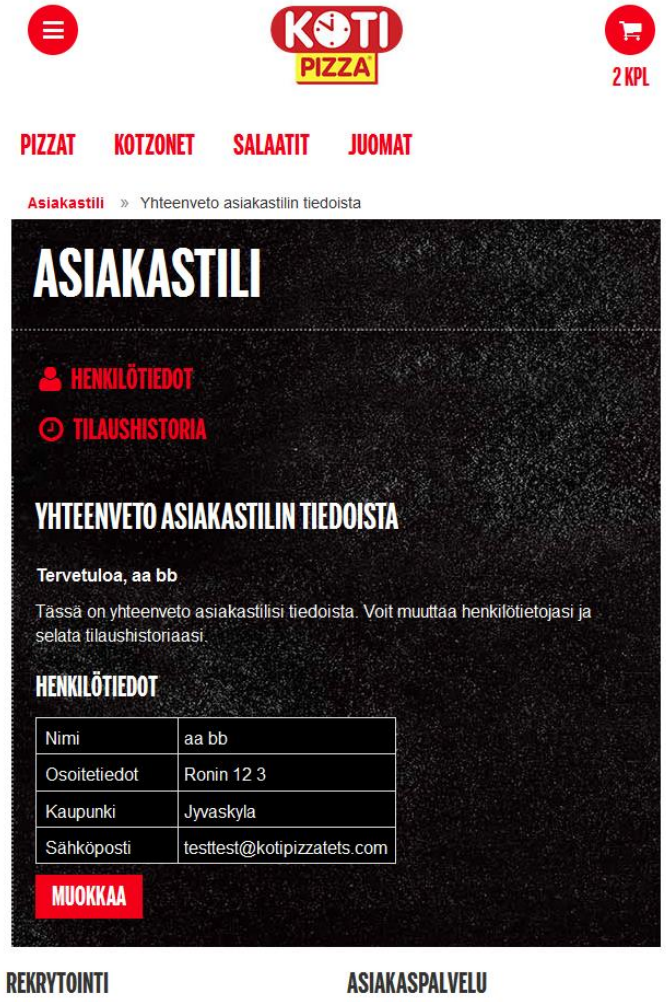

Figure 5. The page based on the example jsp file with the page width 620px. New style sheet from media queries is used.

2.7 Usage of JavaScript

Unfortunately not all of the browsers support the CSS3 media queries. Web page caniuse (http://caniuse.com/) can be used to check if the method is supported by every browser. Nowadays only Internet Explorer version 8 is not supporting media query. Another method that can be used instead of media query is JavaScript.

JavaScript is an object-oriented language. It is used as scripting language for Web pages. When the content and structure of the web page are created by

HTML and the style of the content is defined by CSS, the JavaScripts can be used to add interactivity to the web page. (JavaScript Introduction)

The example below shows how to use JQuery to detect the browser width. The width of the window is assigned to the variable ***var newWindowWidth***. Then in two if loops the variable is compared to the breakpoint width which is 600. If the condition will be met by if loop correct style sheet will be used. In this example two separate CSS files must be created. (Knight 2011)

```
<script type="text/javascript"
        src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.4/jquery.min.js">
</script>

<script type="text/javascript">
        $(document).ready(function(){
                $(window).bind("resize", resizeWindow);
                function resizeWindow(e){
                        var newWindowWidth = $(window).width();

                        if(newWindowWidth < 600){
                                $("link[rel=stylesheet]").attr({href : "mobile.css"});
                        }

                        else if(newWindowWidth > 600){
                                $("link[rel=stylesheet]").attr({href : "style.css"});
                        }
                }
        });
</script>
```

However, for creating responsive UI, if it possible, it is better to use media queries because they are the pure CSS solution.

2.8 Images

It is possible to create page which text will reflows depending on the size of the changing flexible container. It is happening because line of the text can be broken in every spot and continued from the new line. The size of the text is not really changing only the spot where it is placed; however, pages usually do not only consist of the text elements. What will happen if the fixed-width components will be placed in the flexible grid?

An image can be dropped in the HTML structure of the page. The snippet of the code can be looking like this shown in example below.

```
<div class="col7">
        <img alt="" src="pizza.png">
</div>
```

Until the column col7 has the width greater than the width of the image layout is looking fine, and the image is placed in borders of the left column (See Figure 6). The image has resolution 610x610 while the width of the column is 683px.

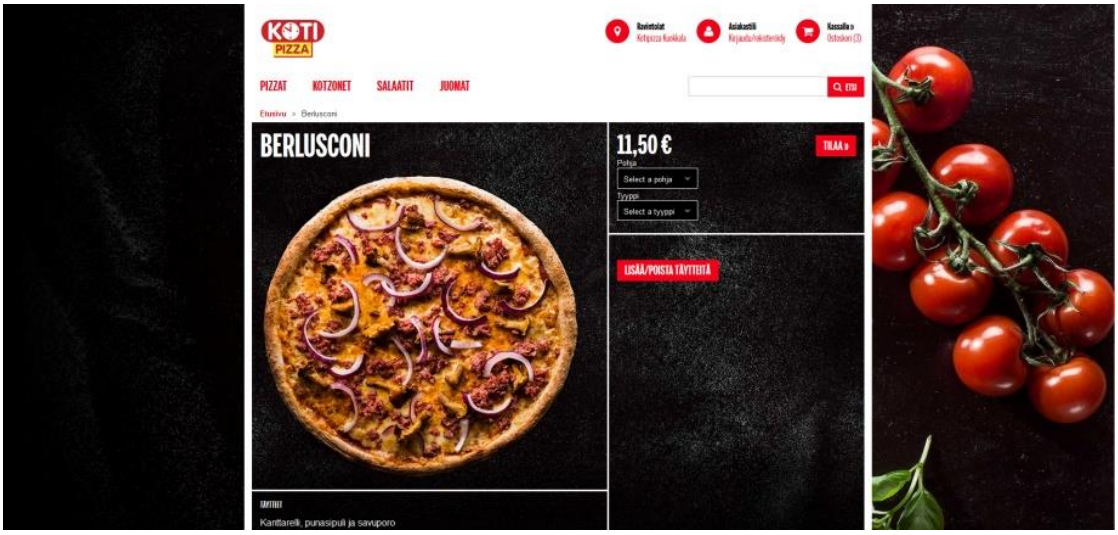

Figure 6. The fixed-size image placed in flexible grid layout, width of the column is bigger than width of the image.

However, if the size of the screen is to be changed to smaller, flexible col7 will automatically resize its width, based on the 58.3333% of whole page, while the width of the fixed-size image will still be the same.

The figure 7 shows resized page to the width 757px. The left column col7 has the width 424px. The image is too big to be placed correctly on the page. It overflows the col7 because there are no constraints that can make the image respond to change of the flexible grid.

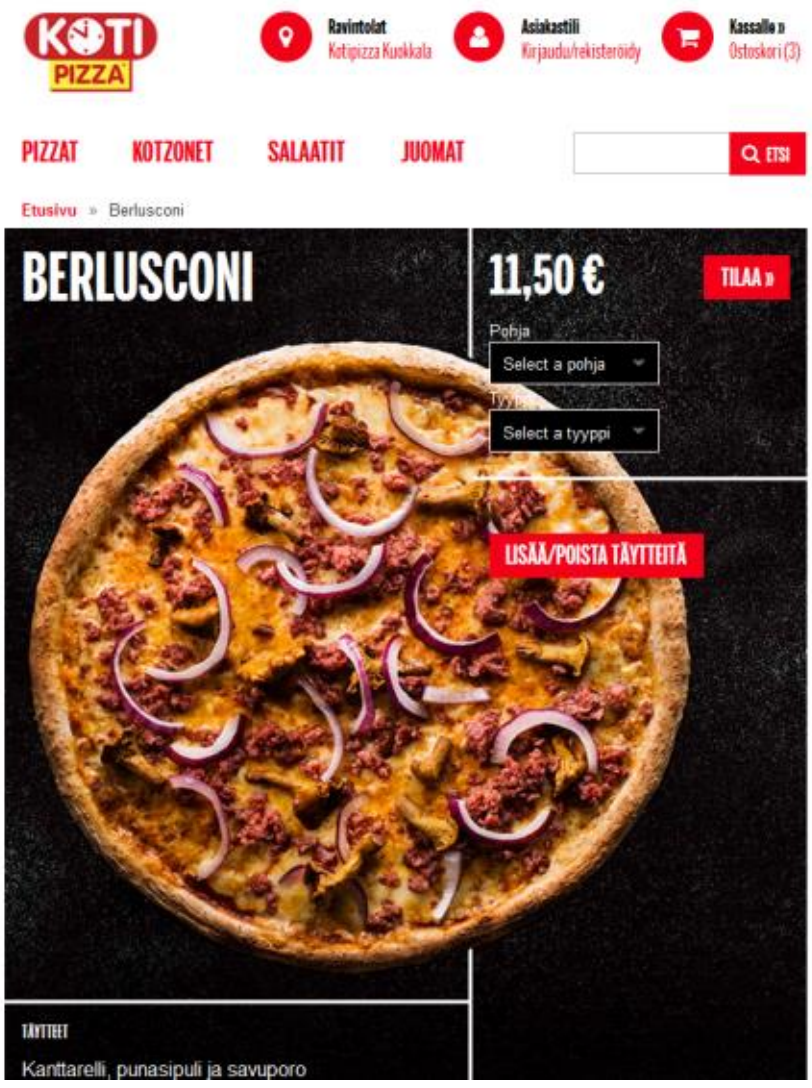

Figure 7. The fixed-size image placed in flexible grid layout, width of the column is smaller than width of the image.

The solution to this problem is to change the fixed size image to the flexible image. It can be achived by using the style ***max-width: 100%*** for the image tag. (Marcotte 2011, 42-47)

```
.img {
    max-width: 100%;
}
```

As long as the image is narrower than the container it will be shown in its normal size, but when it will be too big for the container the image width will be forced to match the width of the container.

The image size will be changed proportionally that mean the height of the image should not be set. This rule will apply to all of the ***img*** elements used on the page. It can be also used to other fixed-size element like video.

Instead of the rule ***width*** which forces the image to always match the width of the container usage of the ***max-width*** requires the width of the image to never be bigger than the width of the container. That means the ***max-width*** rule provides that the image will not be up-scale. This is a really important difference if small images likes thumbnails will be used.

Figure 8 shows the same page as Figure 7 but with a flexible image.

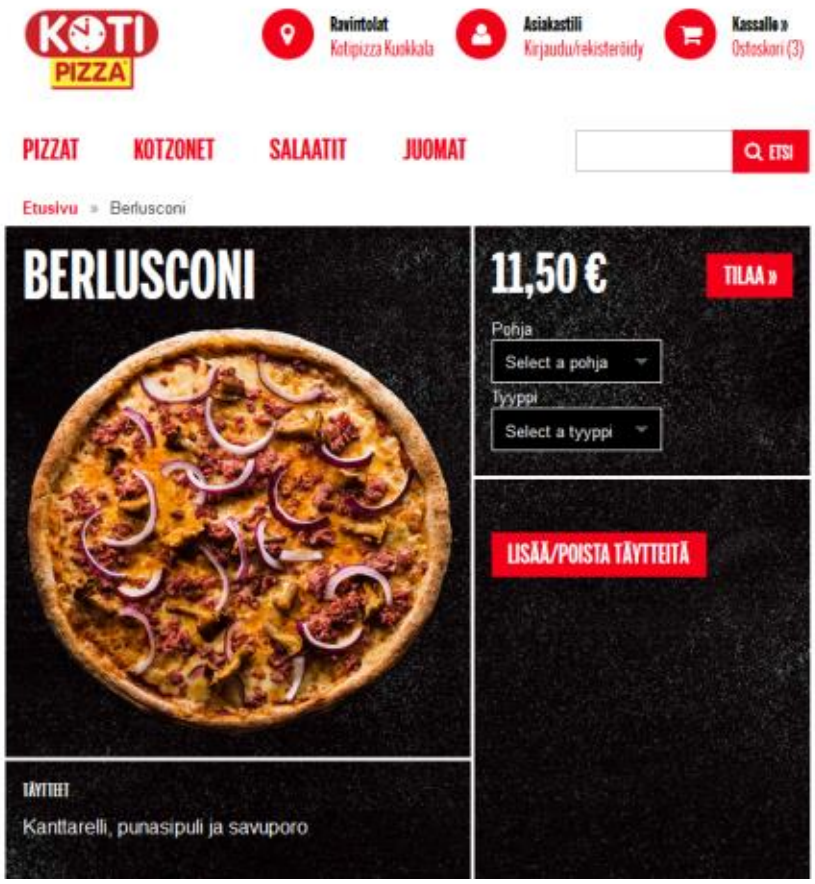

Figure 8. The flexible image placed in flexible grid layout

2.9 Showing and hiding content

Using the techniques shown in previous subchapters, it is possible to resize and adjust all of the contents to make them fit to the smaller pages. However showing every piece of content available at the big screen on the smaller screen is not always a good idea.

Mobile users usually want to find quickly information that they looking for, without long scrolling down the page. When creating pages for phones and smaller devices the most important matter is to keep it as simple as is possible. It is because the lack of space that can be used to show the content and the internet connections is usually slow. The best working layout for smaller screen device is one column layout.

To hide elements that not need to be shown on smaller devices following style can be used on the html block element ***display: none***. This style can be used also on the elements that should be hidden on the desktop devices and be only shown on the smaller one. (Knight 2011)

As an example the basic page with two sidebar on the left and right side, and the main content at the middle of the page, can be considered (See Figure 9).

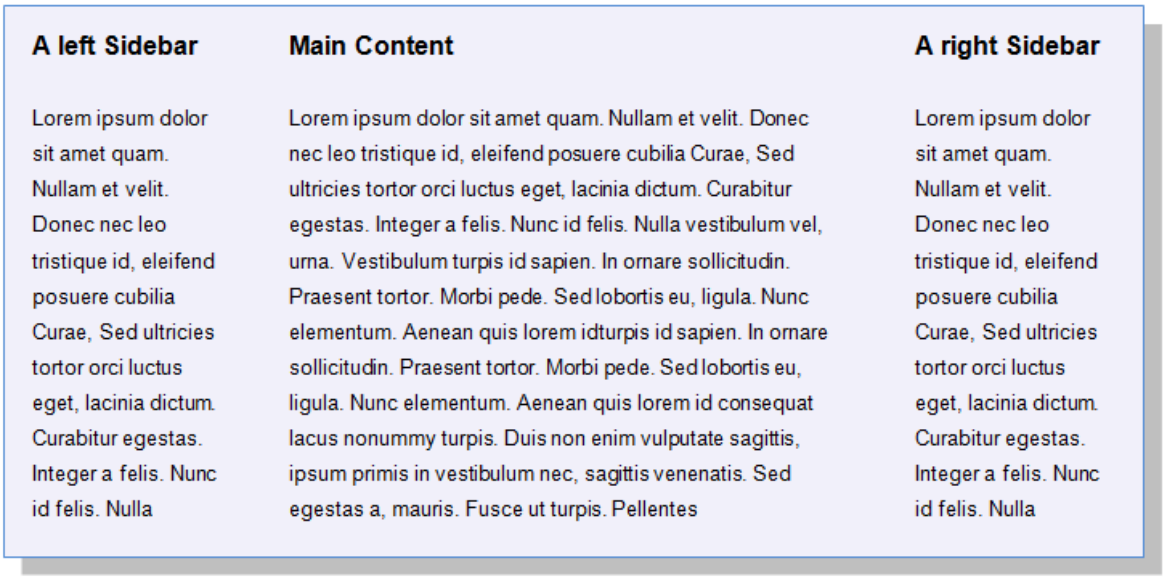

Figure 9. Page with two sidebar on the left and right side, and the main content at the middle of the page. Desktop View.

When the page is viewed on the wider screen, all of the components will be shown; however, on the smaller screen the sidebars have the style display none, which means that they will not be shown. Instead of it the links to these sidebars are shown in the top of the main content (Figure 10). Links will be hidden on the wider screen when the sidebars are available.

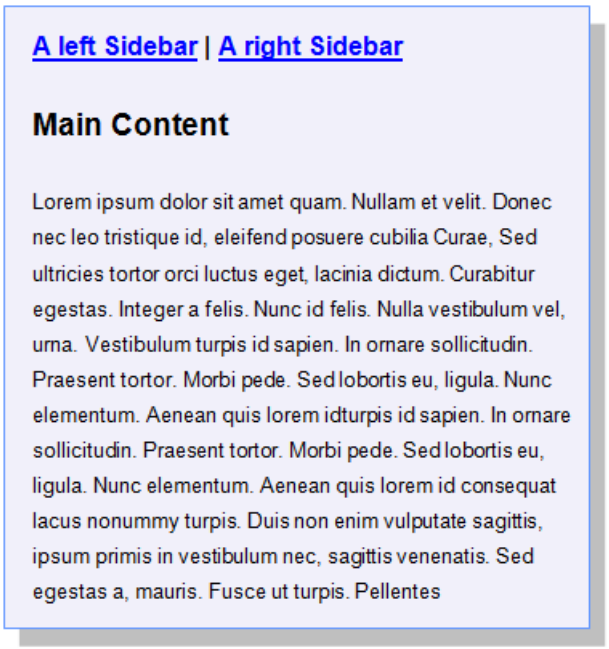

Figure 10. The same page as on Figure 9; however, viewed on the smaller device. Two sidebars from the wider screen are hidden and replaced by links.

The snippet of the CSS file responsible for the style from the example above could be looking like this:

```
.content { width: 54%; float: left; margin-left: 3%; }
.sidebar-left { width: 20%; float: left; margin-left: 3%; }
.sidebar-right { width: 54%; float: left; margin-left: 3%; }
.sidebar-navigation { display: none; }
@media (max-width: 600px) {
    .sidebar-left { display: none; }
    .sidebar-right { display: none; }
    .sidebar-navigation { display: block; }
}
```

3 IBM WEBSPHERE COMMERCE

Electronic commerce (e-commerce) includes processes that are connected with selling and buying products or services and their distribution and marketing using electronic systems. Nowadays e-commerce mostly uses the Internet network.

IBM WebSphere Commerce is a software platform that provides all of the functionality connected with e-commerce. Usage of the IBM WebSphere Commerce can be beneficial for all kind of companies, the small one but also for the large, in every type of the industry. (WebSphere Commerce product overview 2014)

3.1 WebSphere Commerce Architecture

The following software components are associated with WebSphere Commerce: Web Server, WebSphere Application Server and WebSphere Commerce Developer. (WebSphere Commerce common architecture 2014)

Figure 11 shows how they are interconnected.

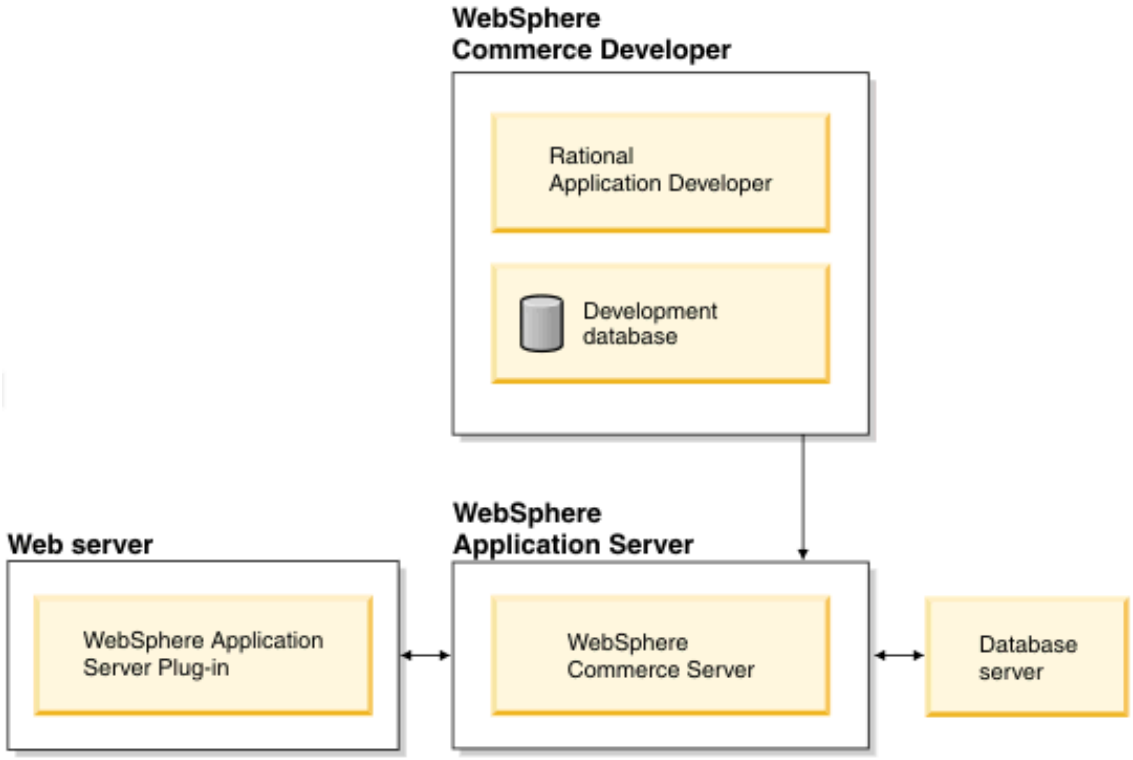

Figure 11. Software components that relate to Websphere Commerce

The incoming HTTP request for e-commerce application is coming at first to the web server. The WebSphere Application Server Plug-in is being used to ensure connections between Web server and WebSphere Application Server. The WebSphere Commerce Server runs inside the WebSphere Application Server, and can use all of the features from application server. The database server stores most of the application data, like product and customer data. (WebSphere Commerce common architecture 2014)

Rational Application Server can be used to execute several tasks such as:

- Creating and customizing storefronts assets like JSP and HTML pages
- Creating and modifying business logic in Java
- Testing code and storefront assets
- Creating and modifying Web services

3.2 WebSphere Commerce Application Layers

WebSphere Commerce Application architecture consists of 7 layers. Each of these layers has the different functionality (WebSphere Commerce application layers 2014)

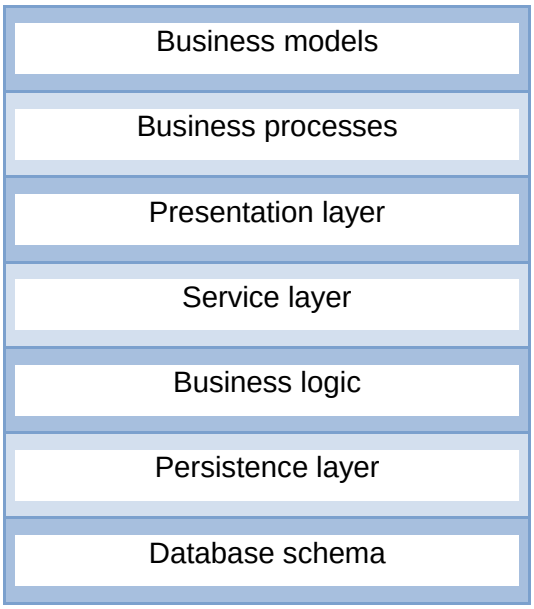

Figure 12. WebSphere Commerce Application Layers

Database schema

It is the bottom layer which stores all data from the WebSphere Commerce Server. Examples of the tables that are stored in database can be Order Table, Member Table, CatEntry Table. (WebSphere Commerce application layers 2014)

Persistence layer

This layer registers the data and operations made using WebSphere system. The layer represents entities used to encapsulate the data-centric logic that is needed to take information from database. They act like an interface between

the business components and the database. (WebSphere Commerce application layers 2014)

Business logic

It contains the actual actions of the WebSphere Commerce Server. In this layer the business rules are implemented using the command pattern. There are two types of commands: controller commands and task commands. Implementation is made independently of the presentation layer. (ibid.)

Service layer

This Layer includes channel independent mechanism that allows accessing WebSphere Commerce business logic. It shows the business logic to the outside world. It supports two transport mechanisms: local Java binding and Web services. (ibid.)

Presentation layer

The task of the layer is to display results. There are two types of presentation layer supported by IBM WebSphere Commerce. The first of it is Web presentation layer, the display is rendered using JSP files. The second type is rich client, for this type presentation is rendered using Eclipse views and editors implemented with SWT components. (ibid.)

Business processes

Business processes show the processes available in WebSphere Commerce. They are divided, according to the business model, into three areas: Administrative processes, starter stores, solutions. (ibid.)

Business models

Business models describe the situation in which WebSphere Commerce products can be used to reach the goals. There are five business models provided by WebSphere Commerce: B2B (Business to business) direct, Consumer direct, Demand chain, Hosting, Supply chain. The sample starter

stores are created based on these business models, and can be developed for users' own needs. (WebSphere Commerce application layers 2014)

3.3 WebSphere Commerce presentation layer

The view layer of the Model-View-Controller (MVC) design pattern is implemented using the JSP. The view layer uses data beans to retrieve and format data from the database. It also decides if the request should be sent to a browser or streamed out as XML. JSP files separate data content from presentation. (WebSphere Commerce presentation layer 2014)

Figure 13 shows how the presentation layer works when the user uses WebSphere Commerce store pages.

1. The user is browsing store pages
2. The servlet is getting the store data from servers using Java beans
3. JSP uses the data to display the information on the store pages
4. JSP interacts with store data via Java beans

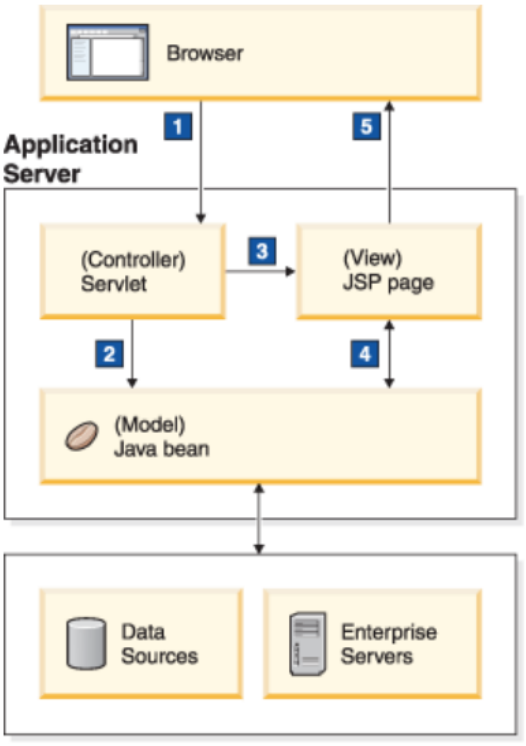

Figure 13. Working of the presentation layer

3.4 Feature Pack 7

In March 2014 Feature Pack 7 was released by IBM. One of the most important enhancements from the User Interface point of view was providing Responsive Web Design for Aurora Starter Store, which is achieved by using fluid layouts and media queries instead of designing pages for each of the screen resolutions. (Highlights of Version 7 Feature Pack 7, 2014)

3.5 IBM Management Center

Management Center framework is used for managing tools that can be used by business users to manage the business and web store. Management Center is created as the graphical Web-based tool. IBM Management Center is a set of the tools that a business user can use to manage the web store. Examples of the business users are Sellers, Marketing and Product Managers. Management Center consists of different components, each of the component is responsible for a different functionality. The following task can be performed using Management Center (Table 1). Site administrator can decide about the Management center roles for the business users. Depending on the role users can use a specified tool. (IBM Management Center for WebSphere Commerce 2014)

Table 1. Usage of the Management Center Tools

Task	Management Center Tool	Examples of actions
Managing catalogs and merchandise	Catalogs Tool	Generating and maintaining master and sales catalog
Managing promotions	Promotions Tool	Generating and maintaining promotions, importing exporting code for the promotions
Managing stores	Store Management Tool	Changing store profile information, like contact information, location
Managing layouts for store pages	Page Layout Tool	Add layout to the page
Creating pages and layouts	Commerce Composer Tool	Creating pages for store, creating layout for pages based on the layout templates

Inside Management Center a new tool can be created by developers for the business user. If the modification is to be done inside any of the tools, Management Center framework needs to communicate with the WebSphere Commerce Server. Different types of services can be used for this communication depending on the need. There are three types of services: configuration (returns configuration data), data (returns business objects) and transaction (create, update and delete business objects).

3.6 Commerce Composer Tool

Commerce Composer Tool is a new tool inside management center. It was introduced in Feature pack 7. Commerce Composer Tool gives more control over the web store and a possibility of creating pages and the layouts to the business users without using IT. Commerce Composer tool holds the library of the layout templates and widgets that can be used as started points in process of creating a new layout for the page. All of the layout template and widgets in Commerce Composer follow a responsive web design pattern. IT developers can prepare and load data to the database, new layout templates

and widgets that later can be used in commerce composer. (Commerce Composer Tool 2014)

Sample use of the Commerce Composer Tool for assigning a new layout to the existing or new page can look as is illustrated in Figure 14.

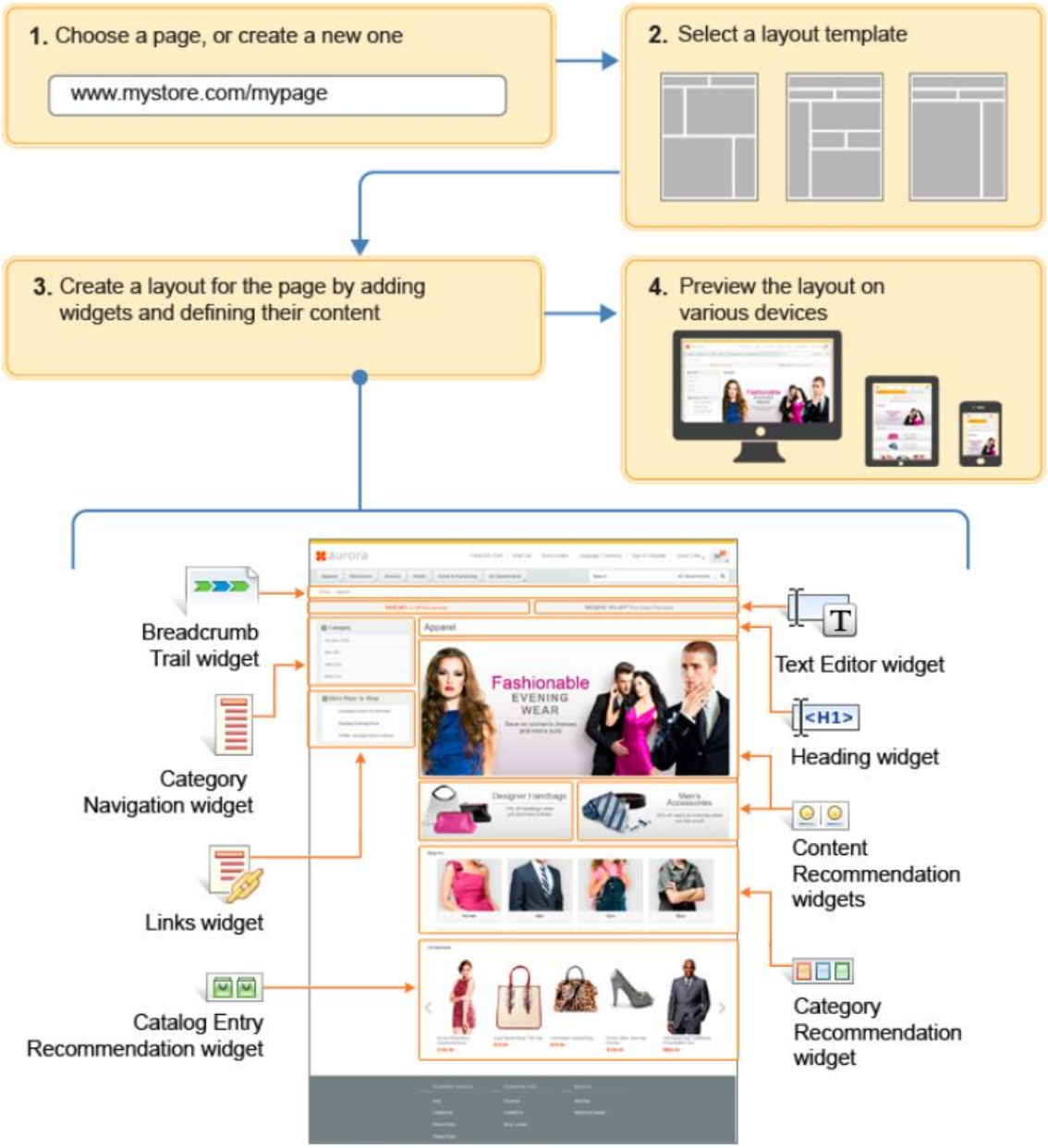

Figure 14. Assigning a new layout to the existing or new page

3.6.1 Pages

In commerce composer the pages are considered as just a URL, without any specific content inside. There are two types of pages: catalog pages (created automatically when new catalog entries are added to master and sales catalogs) and content pages (every other page inside site for example: Contact Us, those pages can be created in commerce composer). (Pages and page creation 2014)

Content page can be created by setting ***URL keyword***. Other data can also be adjusted such as page title or Meta description. The URL should be unique, a simple name without any parameters. Those pages can be viewed by every customer of the web store.

3.6.2 Layout

Layout is responsible for defining and presenting the content of the page. Layout consists of the widgets that have the store content. To use a layout it need to be assigned to the store page. (Layouts, layout template, and default layouts 2014)

A layout template is a base for creating a new layout. It is constructed by a specific setup of the slots (See Figure 15).

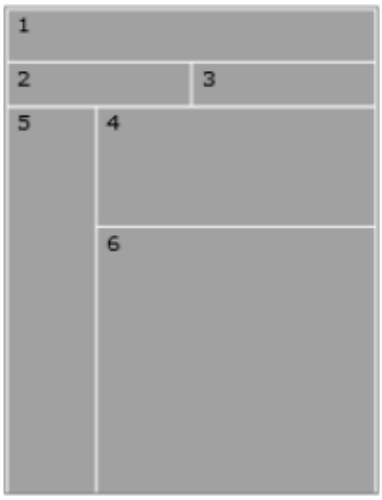

Figure 15. Example of the layout template, numbers present the slot where the widgets can be added.

To create a new layout business user needs to choose the layout template and then place the widgets in to suitable slots. There are three types of the layout template: Desktop layout templates for pages viewed on desktop devices, Mobile layout templates for pages viewed on mobile devices, Responsive layout templates for pages viewed on any devices.

3.6.3 Widgets

A widget is a construction to show store content. Widgets define and retrieve their content in different ways, it can take place automatically or it needs to be set by business user. It is possible to set the properties of the some widgets such as orientation (vertical, horizontal) or initial view (grid, list). (Widgets for Commerce Composer 2014)

Some of the widgets can be used only on the concrete pages and when it is used on a different page an error or empty widget can occur. An example of the page-dependent widget is Full Image widget which can be placed only on catalog entry pages and will retrieve an image of the catalog entry.

An example of a widget that can be used on every page is Breadcrumb Trail widget (Figure 16), which displays the present location of the page in the site hierarchy.

Figure 16. Breadcrumb Trail widget

3.7 Aurora Starter Store

Aurora Starter Store is a sample online store provided by IBM. The usage of the Starter Store makes the process of the creation of an online store faster and easier. The Store can be used as a base for a new customized store. The

Store includes all of the most needed functionalities, which allows proceeding from an ordering process through a buying process. (Starter stores 2014)

3.8 Breakpoints

By default there are three page range breakpoints in Aurora (Table 2). Those breakpoints are based on the common screen size. Of course it is impossible to set the common screen size for each of the devices. The RWD-B range is used as a primary throughout all pages at a web store. If a special style needs to be set for other ranges, the media queries are used. (Page range breakpoints in the Aurora starter store 2014)

Table 2. Aurora default page range breakpoints

Device	Range breakpoint	Page range
Desktop	RWD-C	1281 CSS pixels and above
Tablet	RWD-B	601 – 1280 CSS pixels
Mobile	RWD-A	600 CSS pixels and below

3.9 Store Pages

There are two types of store pages in Aurora Starter Store. The first type of them are responsive store pages; those can rearrange the content according to the size of the screen device. The second type consists of device-specific pages. Device detection framework is used to enable transition between both of these types. Using this framework the devices are identified based on their user agent string. Examples of the device-specific pages are My Account pages and Shopping Cart pages; it means that to show the view of the page, different JSP files will be used, according to the size of the device (Responsive and device-specific store pages 2014)

4 CASE STUDY KOTIPIZZA

The following chapter shows some of the changes that have been made to customize Aurora starter Store to become Kotipizza Store. Across all of the pages minor changes have been made, e.g. changing padding and margins of the pages, also all of the content show on the pages has been modified. Additionally, some major changes have been done to ensure that the pages will be working responsively for the content of the Kotipizza.

The main modification that has been made was making device-specific pages to work as responsive pages. It was accomplished by changing the path to the JSP file defining the layout to always use the same file, even if the pages are used on smaller devices and making this file working responsive.

4.1 Header and Footer

Header and footer are the fields that are separated from the main content of the page. There are the same fields on each of the pages in the store. The header consists of the logo of the web store, catalog browsing menu, category navigation menu and mini shopping cart, links to account and store locator pages. The footer consists of the logo of the web store and the links to the other pages existing in the web store. Both, header and footer change responsively according to the page size. (Header and Footer 2014)

The example below shows the snippet of the JSP file with the main containers of the page.

```
<div id="page">
    <div id="headerWrapper"><!-- Content --></div>
    <div id="rowContainer"><!-- Content --></div>
    <div id="footerWrapper"><!-- Content --></div>
</div>
```

Header

In the following part the header will be analyzed comprehensively. It consists of two rows. The elements located in these rows are shown or hidden depending on the media queries and breakpoints. Figure 17 and Figure 18 show how the header looks with the page width bigger and smaller than 750px.

Figure 17. Header of the page with the width of the page 1200px.

In the desktop version of the header in the first row there is the logo of the store and the links and buttons to other pages in the store.

Figure 18. Header of the page with the width of the page 689px.

In the mobile version, the logo of the store is placed in the middle of the page. Only the shopping cart button is shown in the right part of the page. The rest of the buttons are hidden and instead of them their functionality is moved to the list that is shown when the button on the left of the page is pressed (See Figure 19).

Figure 19. Header of the page, the width of the page being 689px. The button on the left site of the header is pressed.

The snippet of the HTML code, responsible for creating the header is illustrated below:

```
<div id="headerRow1">
    <div class="headerRow1-left">
        <div class="headerRow1-left-mobileContent"><!-- Content --></div>
        <div class="headerRow1-left-desktopContent"><!-- Content --></div>
    </div>
    <div class="headerRow1-middle">
        <div class="headerRow1-middle-mobileContent"><!-- Content --></div>
    </div>
    <div class="headerRow1-right">
        <%out.flush();%>
        <c:import url="${env_jspStoreDir}
                Widgets/MiniShopCartDisplay/MiniShopCartDisplayRefresh.jsp"/>
        <%out.flush();%>
        <div class="headerRow1-right-desktopOnlyContent"><!-- Content --></div>
    </div>
```

```
</div>
<div id="headerRow2">
    <!-- Content -->
</div>
```

The snippet of the CSS file, which sets the style to the HTML page, is shown below. When the width of the screen is smaller than 750px *.headerRow1-left-mobileContent*, *.headerRow1-middle-mobileContent* will have style ***display: block*** that means they will be shown and the *.headerRow1-left-desktopContent, .headerRow1-right-desktopOnlyContent* will have style ***display: none*** that means they will be hidden. As the result of the usage of both styles: ***display-none*** and ***display-block*** the header changes from the desktop version to the mobile version according to the breakpoints set in the media query.

```
.headerRow1-left-mobileContent { display: none; }
@media (max-width: 750px) {
    .headerRow1-left-mobileContent { display: block; }
}
.headerRow1-left-desktopContent { display: block; }
@media (max-width: 750px) {
    .headerRow1-left-desktopContent { display: none; }
}
.headerRow1-middle-mobileContent { display: none; }
@media (max-width: 750px) {
    .headerRow1-middle-mobileContent { display: block; }
}
.headerRow1-right-desktopOnlyContent { display: block; }
@media (max-width: 750px) {
    .headerRow1-right-desktopOnlyContent { display: none; }
}
```

The mechanisms used to show and hide elements in the second row are the same as the ones used in first row.

4.2 Page Template

The following snippet of the CSS file demonstrates the rules that apply to all of the ***page*** elements from web pages. The pages will work responsively (resize the width) only for the width of the screen device smaller than 984px. Used media queries provide that the content of the page will not become too wide.

```
div#page {
    width: 100%;
    height: 100%;
    position: relative;
    background: #FFFFFF;
}
@media only screen and (min-width: 984px) {
    div#page {
        width: 960px;
        margin-left: auto;
        margin-right: auto;
    }
}
@media only screen and (min-width: 1281px) {
    div#page {
        width: 1200px;
        margin-left: auto;
        margin-right: auto;
    }
}
```

4.3 Category Page – New layout template

The new layout template was created for the needs of the category page. Snippets below show container JSP file which defines this layout template. It consists of three rows; each of these rows has the width of 100%. The columns inside rows represent the configurable slots. The line `<wcpgl:widgetImport slotId="1" />` is responsible for importing widgets to the defined slots. The id of the slot must be valid with those which are set in the layout template in Commerce Composer.

```
<div class="rowContainer" id="container_${pageDesign.layoutID}">
        <%-- First row --%>
        <div class="row margin-true" id="breadcrumb">
                <%-- First column --%>
                <div class="col12 acol12" data-slot-id="1">
                        <wcpgl:widgetImport slotId="1" />
                </div>
        </div>
        <%-- Second row --%>
        <div class="row">
                <%-- First column --%>
                <div class="col12 acol12" data-slot-id="2">
                        <wcpgl:widgetImport slotId="2" />
                </div>
        </div>
        <%-- Third row --%>
        <div class="row">
                <%-- First column --%>
                <div class="col12 acol12"  data-slot-id="3">
                        <wcpgl:widgetImport slotId="3" />
                </div>
        </div>
</div>
```

Before layout template can be used in Commerce Composer, it needs to be register in Commerce Composer framework and the store needs to be subscribed to this layout template, in order to do this data load utility is used. Data load input file needs to be modified. There are four of them (Registering a Commerce Composer layout template 2014):

- registationWidgetDef.csv – specifies the information for registering a layout template container, such as widget vendor or path to the JSP file
- subscribeWidgetDef.csv – subscribes a store to a layout template container
- template.csv – sets information about template, like layout template name and description, that will be shown in commerce composer
- slotdefinition.csv – loads the slot definition information, this file is responsible for a view of the layout the template shows in the commerce composer

When all files are adjusted and the data load utility has been run, it is possible to use the new layout template within the commerce composer.

Figure 20 shows the layout template created for the Kotipizza category page. There are three widgets placed in the slots. Breadcrumb Trail Widget, Heading Widgets (display the name of the category) and Catalog entry list Widget (display the list of the catalog entries).

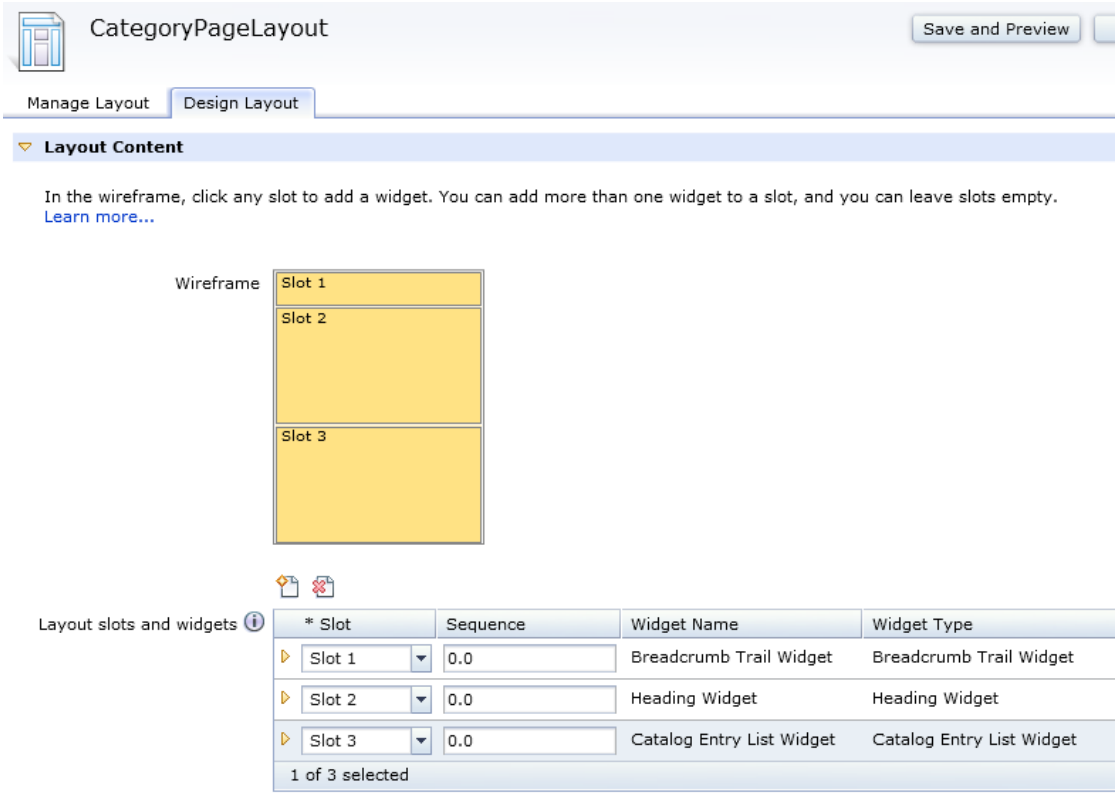

Figure 20. Layout template for Kotipizza Category Page

4.4 Checkout Process – new breakpoints

The shopping cart page is the example of the page for which the default Aurora starter store breakpoints do not work very well (Figure 21).

That is why the new class col6-descom was created. It has the word col6 included the same naming conventions as in fluid grid. Col6-descom has, as col6, width set to 50%. The difference is that the other breakpoints will be used for the class col6-descom. If the width of the page is smaller than 780px, the new width will be assigned to the col6-descom. The shopping cart page will no longer consist of the two columns like it was with the width bigger than 780 (Figure 22). Instead of it, the columns are placed one below the other (Figure 23).

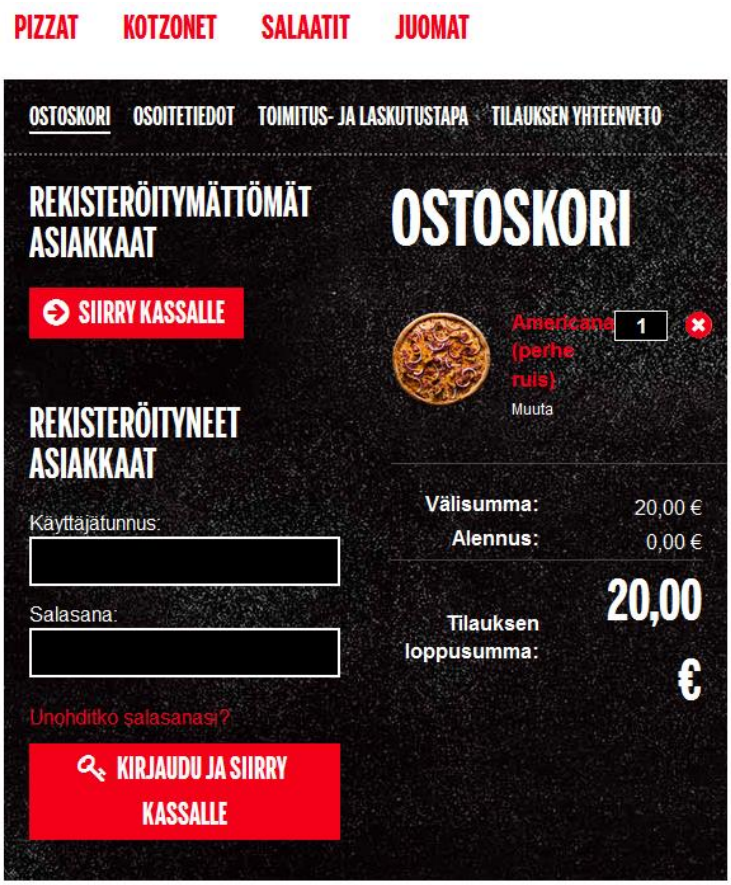

Figure 21. Shopping Cart Page with default break points. The page width is 600px.

The following snippet of the CSS file demonstrates how class col6-descom and media query are defined. The new breakpoint is set to the value of 780px

```
.col6-descom {
        width: 50%;
        padding: 0;
}


@media (max-width: 780px) {
        .col6-descom {
                width: 100%;
        }
}
```

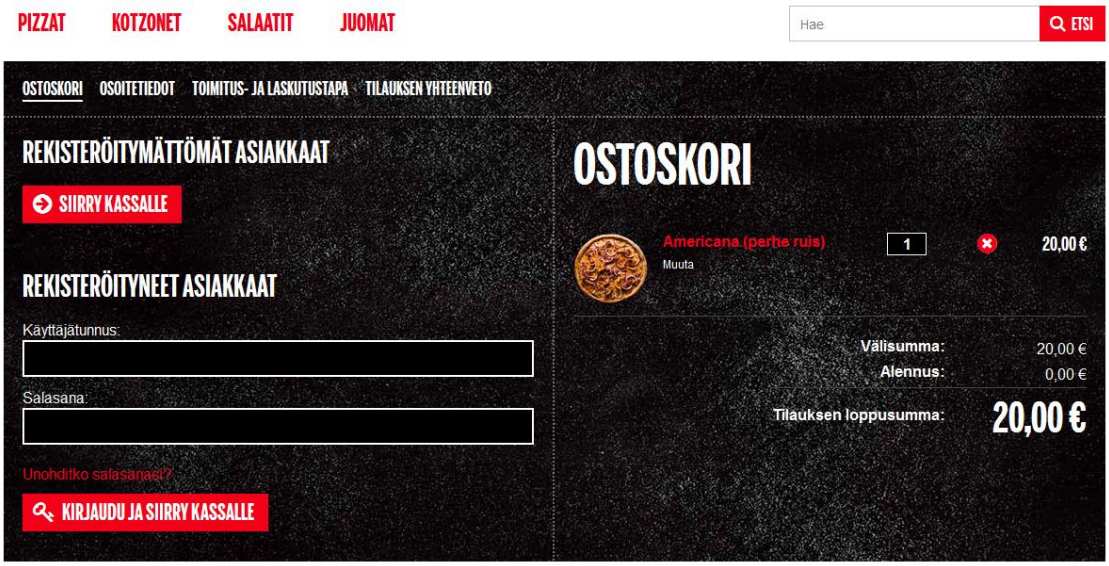

Figure 22. Shopping Cart Page with custom break points. The page width is 600px.

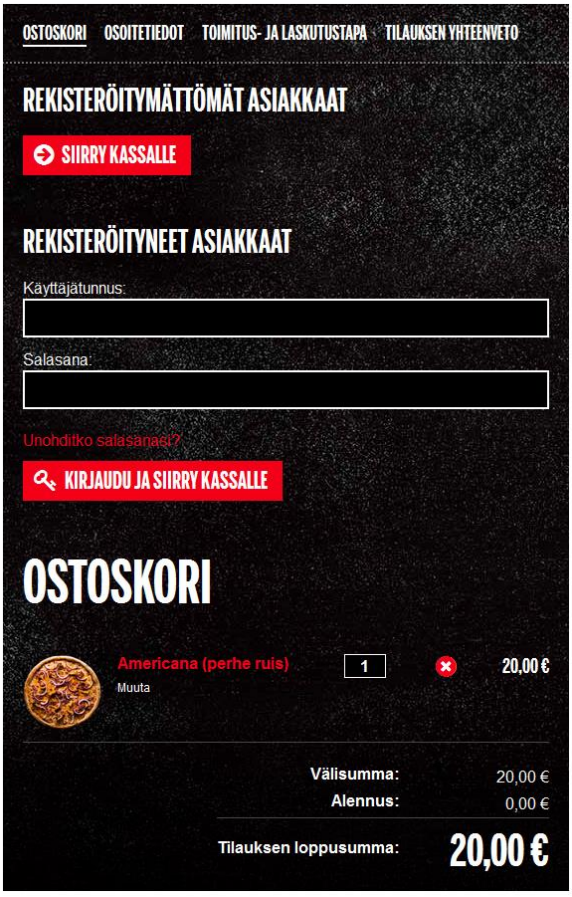

Figure 23. Shopping Cart Page with custom break points. Width of the page is 600px.

5 CONCLUSION

The purpose of the thesis was to create responsive UI for a web store. It was accomplished by using fluid grids, fluid images and media queries. The breakpoints for each of the pages were investigated separately. They were set depending on the contents of the page. Good practice of showing and hiding elements on devices with different resolution was used to provide the best usefulness of the web store. All of those rules ensure that the web site looks as good in mobile devices as in the desktop ones without duplicating the code. Design of the web store was created to make the functionality of the store easy and logical for all of the users.

IBM WebSphere Commerce framework was used to create the web store. Use case that it is shown in thesis was created based on the Aurora Starter Store. Almost all of the pages needed for the store were already implemented. There were used as foundation for the store.

From the web designing point of view changing the user interface to act responsively was not hard. The challenge was to get to know IBM WebSphere Commerce framework and Aurora Starter Store. The file structure of the store is complicated. Sometimes it was hard to know which JSP file is responsive for which view. Also, inside JSP files widgets and new JSPF (JSP Fragments) files were imported, which makes it even more complicated.

Working with IBM WebSphere Commerce was a new useful experience. The project gives opportunity to extend web developing skills. Another profitable aspect of this project was working in the team, where everyone needs to be able to communicate to everyone else. Finally, all of the goals of the thesis were achieved.

REFERENCES

Commerce Composer Tool. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 9 May 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/concepts/cpzpagecomptool.htm

Header and Footer. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 5 May 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.aurora-starterstore.doc/refs/rsmaurorasa_atoverall.htm

Highlights of Version 7 Feature Pack 7. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 27 April 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.admin.doc/concepts/cwnFEP7summary.htm

IBM Management Center for WebSphere Commerce. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 9 May 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/concepts/ctfcmc.htm

JavaScript Introduction. Page on w3schools, Accessed 4 May 2014. Retrieved from http://www.w3schools.com/js/js_intro.asp

Knight, K. 2009. Fixed vs. Fluid vs. Elastic Layout: What's The Right One For You? Accessed on 27 April 2014. Retrieved from http://www.smashingmagazine.com/2009/06/02/fixed-vs-fluid-vs-elastic-layout-whats-the-right-one-for-you/

Knight, K. 2011. Responsive Web Design: What It Is and How To Use It Accessed on 4 May 2014. Retrieved from http://www.smashingmagazine.com/2011/01/12/guidelines-for-responsive-web-design/

Kyrnin, J. How to Use Multiple CSS Classes on a Single Element Accessed on 27 April 2014. Retrieved from http://webdesign.about.com/od/css/qt/tipcssmulticlas.htm

Layouts, layout template, and default layouts. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 9 May 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/concepts/cpztemplates.htm

Marcotte, E. 2011. Responsive Web Design. New York, New York: A Book Apart

Media Queries. 2012, Page on W3C Recommendation, Accessed 27 April 2014. Retrieved from http://www.w3.org/TR/css3-mediaqueries/#media0.

Pages and page creation. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 9 May 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/concepts/cpzpages.htm

Page range breakpoints in the Aurora starter store. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 27 April 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.aurora-starterstore.doc/concepts/csmaurorarwdpagebounds.htm

Registering a Commerce Composer layout template. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 27 April 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.pagecomposerframework.doc/tasks/tpzlayouttemplatecreatereg.htm

Responsive and device-specific store pages. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 27 April 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.aurora-starterstore.doc/concepts/csmaurorarwdtransitions.htm

Responsive Web Design (RWD). 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 27 April 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.aurora-starterstore.doc/concepts/csmaurorarwd.htm

Responsive Web Design (RWD) framework. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 27 April 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.aurora-starterstore.doc/concepts/csmaurorarwdframework.htm

Starter stores. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 27 April 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.starterstores.doc/concepts/csmStarterStores.htm

WebSphere Commerce application layers. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 27 April 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.developer.doc/concepts/csdapplication.htm

WebSphere Commerce common architecture. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 27 April 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.developer.doc/concepts/csdsoftwarecomp.htm

WebSphere Commerce presentation layer. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 5 May 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.developer.doc/concepts/csdviewlayer.htm

WebSphere Commerce product overview. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 9 May 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.admin.doc/concepts/covoverall.htm

Widgets for Commerce Composer. 2014. Page on IBM Info Center - WebSphere Commerce Version 7.0.0.7 Accessed 9 May 2014. Retrieved from http://pic.dhe.ibm.com/infocenter/wchelp/v7r0m0/topic/com.ibm.commerce.management-center.doc/concepts/cpzwidgetcont.htm

Van Gemert, V. 2013. Logical Breakpoints For Your Responsive Design. Accessed on 27 April 2014. Retrieved from http://www.smashingmagazine.com/2013/03/01/logical-breakpoints-responsive-design/