Responsive Application Design – Websites to Cross Platform Applications

Gerald Hauser

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Interactive Media

in Hagenberg

im September 2014

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Hagenberg, September 19, 2014

Gerald Hauser

Contents

Abstract

The mobile sector has been increasing enormously within the last few years. New techniques rises that enable developers to deal with the problems that appear with the large amount of different devices, screen form-factors and platforms the devices are running on. Responsive and adaptive designs have prevailed in development of web applications and websites, giving the developer the opportunity to design web applications that adjust to the screen form-factor of the device it is running on. Depending on the requirements of the application, a web application is not always the best solution. Sometimes native applications are a better choice. A convenient characteristic of well designed web applications is that they are running independent of the operating system inside the browser, either mobile or at the desktop. To achieve this in native application development, cross platform frameworks can be used. With the possibility of cross platform frameworks source code is written only once and compiled to the platforms the application should run on. Within this thesis the concept of *Responsive Web Design* is established as well as the possibilities of *cross platform* development. The practical approach of the thesis deals with the implementation of the most useful techniques from Responsive Web Design within a specific cross platform framework, *LibGDX*. In the conclusion the proposal is made to use a more generic expression than Responsive Web Design: *Responsive Application Design*.

Chapter 1

Introduction

Throughout the last years the mobile sector has been increasing very fast. Different mobile devices with diverse platforms and screen form-factors overflow the global market. For a developer of native mobile applications it is difficult to cover all the different types of mobile devices from smartphones to tablets, that run on different operating systems. *Responsive Web Design*, a technique of web design, that has been rising over the past view years, gives the developer the possibility to deal with different screen form-factors when designing a web-application or website. The platform independence of websites and web-applications running in the browser, either mobile or at the desktop, is a convenient feature. But such web-applications are not always the best way in practice, depending on the requirements of the application e.g. access to any hardware features of a mobile device, or time critical operations that can not be done using a low bandwidth. In this cases native applications can be a better choice. The use of a cross platform framework, for native application development, is a possibility, to develop applications that are running on more than one platform natively. So if a native application is required that has to run on several platforms, why not combining the approach of Responsive Web Design with the use of a cross platform framework? The combination of these two approaches gives the possibility to deal with the problematic of various screen form-factors as well as the problematic of different platforms. Within section 1.1 the subject of the thesis is explained. The problem statement is pointed out in section 1.2. To put it in a nutshell, the hypothesis is given in section 1.3. The structure of the thesis is given in section 1.4.

1.1 Subject

The reason for this thesis is to examine if it is possible to adopt and adapt principles of Responsive Web Design, to use them in combination with a specific cross platform framework in order to develop natively running mobile

applications. Furthermore, if the possibility of using principles of Responsive Web Design in application development is given, it can be investigated if the terminology of Responsive Application Design is fitting better than Responsive Web Design, because it covers a larger field in the area of application development.

1.2 Problem Statement

There are several problems when thinking of mobile application development. One specific problem is is to deal with an enormous number of screen form-factors. Manufacturers of mobile devices, especially of Android devices, produce their products in many different sizes, with different displays, resolutions, and aspect ratios. Application-users expect any application to be presented in a proper way, and not to be stretched or cropped on displays with other screen form-factors than they were designed for. It is in the interest of the developer to represent his application to the user accordingly. Another problem is the diversity of platforms that are available. Beside a variety of desktop platforms, e.g. Mac OS X, Windows, and Linux, there are also several mobile platforms, e.g. Android, iOS, and Windows Phone, available at the market nowadays. When there is the need to develop a native application, the implementation can be done for each single platform individually, or using the possibility of cross platform frameworks to write code once and compile it for each platform the application should run on.

1.3 Hypothesis

> Adopting techniques from Responsive Web Design to use them in cross platform development overcomes the problematic of multiple screen form-factors, and provides the possibility to develop native applications for various platforms.

1.4 Structure

This thesis is subdivided in three main chapters. Chapter 2 forms the theoretical foundation of the thesis to give an overview of the state of the art. It consists of the growth of the mobile sector in section 2.1, the different approaches to handle screen form-factors in section 2.2 , introduces Responsive Web Design (RWD) in section 2.3, and cross platform development in section 2.4. Finally section 2.5 draws a conclusion of chapter 2.

The practical approach shown in chapter 3 demonstrates the possibility to develop a cross platform application with responsive characteristics. Section 3.1 gives the purpose of the sample application. The cooperation with

APEX gaming technology GmbH is explained in section 3.2. How the selection of the cross platform framework took place is described in section 3.3. Section 3.4 includes actual application, with the description of the mathematical notations, the architecture and the implementation. The conclusion in section 3.5 forms the end of the chapter.

Chapter 4, finally shows the conclusions of the thesis. The findings of the thesis are summarized. The chapter also proposes areas of further research.

Chapter 2

Theoretical Foundation

This chapter brings closer insight on the topic *Responsive Web Design* (RWD) and *Cross Platform Development* (CPD). It particularly deals with the subjects of growth of the mobile sector, different approaches to handle screen form-factors, Responsive Web Design (RWD) and cross platform development. Section 2.5 contains the conclusion of this chapter, that summarizes the main points again.

2.1 Growth of the Mobile Sector

Within this section definitions and delimitations of the term smartphone are explained. Further the evolution from mobile phones to smartphones is stated and the the latest trends in device shipments 2008 – 2015 are shown as well as a summary of different screen form-factors.

2.1.1 Definitions and Delimitations

The term *smartphone* is defined as a mobile device that provides advanced capabilities beyond a typical *mobile phone* [58]. Early smartphones typically combined the features of a mobile phone with those of another popular consumer device, such as a *PDA*[1], a *media player*, a *digital camera*, or a *GPS*[2] navigation unit. Modern smartphones include all of those features. The list of features a modern smartphone has includes mostly a touchscreen and an accelerometer, web browsing, Wi-Fi, and third-party applications and accessories [58]. Nowadays the term *feature phone* defines any mobile phone that is not a smartphone [57]. Feature phones differ from smartphones in their proprietary operating system firmware, and their rare support of third-party software, e.g. via closed platform and operating system [57], whereas smartphones run accessible operating system software that provides a stan-

[1] PDA: Personal Digital Assistant
[2] GPS: Global Positioning System

dardized application programming interface (API) for developers [58]. A tablet computer or tablets are general-purpose computer contained in a single panel. Just like modern smartphones, tablets also run complete operating system software with standardized interface and platform for application developers. All in all modern tablets often have the same features as modern smartphones (sensors, cameras, microphone, accelerometer, touchscreen, accessories). Tablets nowadays are mostly operated by fingers, and a stylus in an option, whereas earlier tablets required a stylus for input [56].

2.1.2 From Mobile Phones to Smartphones

Taking a look back in history there were already device concepts that combines telephony and computing in 1973, some of them already offered for sale as early as in 1993. A first prototype of a device that incorporated PDA features was developed by IBM[3] in 1992 and demonstrated at the COMDEX computer industry trade show that year. With the name *Simon Personal Communicator* a refined version of the product was designed by IBM. BellSouth Cellular Corp. retailed *The Simon* from August 1994 to February 1995 for $899 per device and sold approximately 50000 units. The Simon was the first device that can properly be called a smartphone, even though that term itself was not coined yet [60]. The first device that was called a smartphone appeared in 1997, when Ericsson described its *GS 88 Penelope* concept as a *Smart Phone* [32]. The first smartphone for mass adoption was released 1999 by the Japanese mobile phone operator *NTT Docomo Inc.* With the mobile Internet service i-mode these smartphones have access to various services such as e-mail, sports results, weather forecast, games, financial services and ticket booking. I-mode used cHTML[4], a subset of HTML[5], in favor of increasing data speed for the devices. NTT Docomo owes it to the i-mode that they have accumulate an estimated 40 million subscribers by the end of 2001. This makes NTT Docomo ranked first in market capitalization in Japan and second globally. With the rise of 3G and new smartphones with advanced network capabilities the supremacy of Docomo wane [10]. Smartphones outside of Japan were still rare. Starting in 2003, devices based on Microsoft's Windows Mobile started to gain high popularity for business use in the U.S. while the same year *Research In Motion* (RIM), nowadays known as BlackBerry [19], had its breakthrough with their first GSM[6] smartphones BlackBerry series 62xx and also the BlackBerry 72xx serie which already had a color screen [35]. Although BlackBerry and Windows Mobile systems had a large lead in the North American market from 2006 to 2007, BlackBerry was known to a more widely distributed

[3] IBM: International Business Machines Corporation
[4] cHTML: Compact HyperText Markup Language
[5] HTML: HyperText Markup Language
[6] GSM: Global System for Mobile Communications, originally Groupe Spéciale Mobile

public of business people and young people. At this time Windows Mobile still had its major proportion in the business sector. At the same time the most popular mobile operating system in Europe was Symbian, largely led by Nokia. Just as BlackBerry and Windows Mobile Nokia was situated in the business sector at the beginning. This changed in 2006, where Nokia changed its courses and started to make smartphones that focuses on entertainment, popularized by the N-series smartphones. With the N95, that had revolutionary multimedia features for its time, Nokia got popular among younger people [54]. In 2007, when Apple released its first iPhone [16] and 2008, when HTC released the first Android smartphone HTC Dream, two new players entered the mobile marked [53]. Over the years, Apple and Android became key-players of the mobile market. Their different approach to reach this is discussed in more detail in the following sections 2.1.3 and 2.1.4.

2.1.3 Apple Inc.

The multinational corporation with the headquarter in Cupertino, California was founded by *Steve Jobs*, *Steve Wozniak*, and *Ronald Wayne* on April 1, 1976 to develop and sell personal computers [17]. With the introduction of the *iPhone* in 2007, as one of the first smartphones that used a large multi-touch-screen for direct finger input instead of a hardware keyboard or stylus, which was state of the art at that time, Apple entered the mobile market. When the first iPhone was launched it did not support to run third-party applications on it. The launch of the *App Store* in 2008 that enables the possibility of developing third-party applications and sell them through the store, is a important key event that changed application development dramatically [14].

App Store

Since the introduction of the App Store and *iPhone OS 2.0* (iOS) in 2008 it is possible to develop third party applications for the iPhone [14]. The App Store became a central platform to commercialize third party applications for iOS devices. Although ambitious software developers had managed to *jailbreak*[7] the iPhone before the release of Apple's official App Store to run self-developed applications on it, distributed through package managers such as Installer.app and Cydia. The App Store allows the users to download and purchase new applications for their device. Developers who distribute their application on the App Store have to obey a large number of restrictions; all applications get reviewed by Apple staff when submitted and get rejected when they do not pass the Apple's technological and design guidelines [13].

[7] Jailbreak: the remove of limitations on Apple's devices running iOS to permit root access to the iOS file system and manager, allowing the download of additional applications, that are unavailable on Apple's official App Store.

Apple benefits from the App Store. As of the time the thesis has been written, Apple takes a 30% fee on revenues for paid applications sold through the store [17], [13].

Financials

The App Store has been a major financial success for Apple. According to current statistics Apple has reached over 40 billion application downloads up to 2013, with more than over 800 000 available applications [15]. Apple's revenue in 2013 of totalled $170 billion makes it the world's second-largest information technology company after Samsung Electronics [51].

Devices

Beside the iPhone, Apple expanded its product range in the mobile sector with the *iPod touch* in 2008 and the *iPad* in 2010 [17], all of them having access to the App Store. This is important because all devices that have access to the App Store also have access to the applications on that store. A developer therefore has to deal with different screen form-factors. In 2012 Apple the iPad was first available with two different specifications – the iPad 4 with more or less the same aspect ratio than its predecessors and the iPad mini which is a large smaller and therefore more handy. Today the iPad is at its fifth generation with the iPad Air and the iPad mini is at its second generation [17]. From the first to the fourth generation of the iPhone its screens aspect ratio stays the same which made it more easy to develop applications for only one aspect ratio. Just with the iPhone 5 the aspect ratio changes. In 2013 the iPhone has reached the more or less the fifth generation with some facelifts. With the fifth generation the iPhone is first available in two completely different versions – the iPhone 5C and the iPhone 5S – using different materials, concepts, hardware and prices to reach a more widespread user base [17]. At the time the thesis was written Apple released the iPhone 6 and iPhone 6 Plus. The two devices differ evidently in their display specifications, with the iPhone 6 that has a 4.7" *Retina HD Display* with 1334×750px resolution at 326ppi[8], whereas the iPhone 6 Plus has a 5.5" *Retina HD Display* with 1920×1080px resolution at 401ppi.

2.1.4 Android Inc.

Android Inc. was founded in Palo Alto, California, in October 2003, by Andy Rubin, Rich Miner, Nick Sears and Chris White to design *smarter mobile devices that are more aware of its owner's location and preferences* [9]. When Google acquired Android Inc. in August 2005 there was not much known

[8] ppi: Pixels per inch

about that company at the time, but many assumed that Google was planning to enter the mobile phone market with this move [31]. At Google, a team led by Rubin started to develop a mobile device platform. The unveiling of the iPhone, a touchscreen-based phone by Apple, on January 9, 2007 had a disruptive effect on the development of Android. At the time, a prototype device codenamed *Sooner* had a closer resemblance to a BlackBerry phone, with no touchscreen, and a physical, QWERTY keyboard. Work immediately began on re-engineering the OS and its prototypes to combine traits of their own designs with an overall experience designed to compete with the iPhone [43]. September the same year, InformationWeek covered an Evalueserve[9] study reporting that Google had submitted several patents in the mobile-phone technologies' area [23]. November 5, 2007, the Open Handset Alliance[10] announced the goal to develop open standards for mobile devices [42]. The same day, Android was unveiled as its first product built on the Linux kernel version 2.6 [42].

Devices and Manufacturers

October 22, 2008, the HTC Dream was released as the first commercially available smartphone running Android [53]. Beside several third party modifications of the user interface (UI) of the original Android operating system Google also launched its Nexus series, to act as their flagship devices with the latest software and hardware, running *Stock-Android* on it[11]. The devices are built by manufacturing partners such as HTC, LG and Asus. 2010 the Nexus One[12] was the first smartphone that runs pure Android. The latest representative of the series are the Nexus 5 phone[13] the Nexus 7 tablet[14] and the Nexus 10 tablet[15]. Devices running Stock-Android are in most cases the first one getting software updates, since third party companies, that deliver a modified UI for Android – e.g. HTC Sense [38], Samsung TouchWiz [64], or Sony UI [18] – have to change new released Android versions to implement their specific UIs [47].

Operating System Release-Philosophy

Android had several updates since 2008, which have incrementally improved the operating system. Major releases are named in alphabetical order after

[9] Evalueserve – a global consulting and research firm

[10] Open Handset Alliance – a consortium of technology companies (Google; device manufacturers: HTC, Sony and Samsung; wireless carriers: Sprint Nextel and T-Mobile; chipset makers: Qualcomm and Texas Instruments)

[11] Stock-Android – a pure version of Android released and updated by Google directly

[12] by HTC

[13] by LG

[14] by Asus

[15] by Samsung

a dessert or sugary treat; for example, version 1.5 Cupcake was followed by 1.6 Donut. The latest version of Android 4.4.4, named KitKat, was released on June 19, 2014 [59].

2.1.5 Device Shipments 2008 – 2015

According to the statistic in figure 2.1 from *Booz & Company Inc.*, a global management consulting firm, the mobile sector (smartphones and tablets) is growing fast. In 2008, global smartphone sales, with 252 million units sold, were overtaken by PC and notebook sales, with 150 million PCs sold and 140 million notebooks sold, added together 290 million units. While the sales of PCs remained at about 150 million units from 2009 to 2012, with a significant drop to 138 million units sold in 2009, the sales of Laptop and smartphone units increased constantly. In 2010, when the first tablets were introduced, 18 million units were sold. In 2011, smartphones sales in addition with tablet sales formed a sale of 427 million units, whereas PC and Laptop sales together remained at 396 million units. Total PC and notebook sales were overtaken by total smartphone and tablet sales the first time. According to the forecasts to 2015 the trend is still unbroken. Sales of smartphone (776 million sold units in 2015) and tablets (445 million sold units in 2015) will increase dramatically whereas the increase of notebook sales, with 464 million units sold in 2015, is rather slow. PC sales are forecasted to increase slightly to about 157 million units [67].

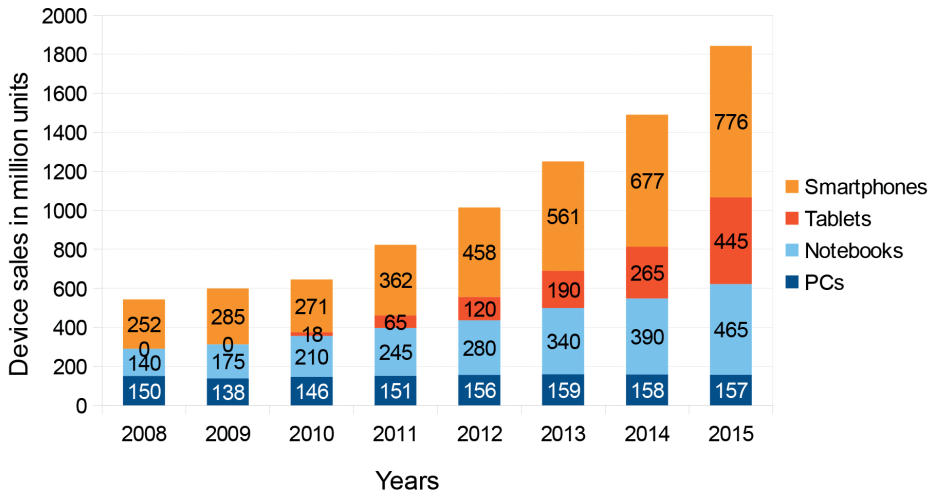

Figure 2.1: Global device sales 2008 – 2012, with forecasts to 2015. Adopted from Booz & Company Inc. [67].

There are two reasons why the mobile sector is growing. The First reason is that smartphones are getting better in use and functionality and also

the hardware specifications improve e.g. better camera, accelerometer or GPS. Another reason is that there is a smartphone for almost every budget, already starting at about 40$ [52] for a contract-free smartphone. The web can be accessed with affordable mobile devices and relatively cheap mobile data plans, a laptop or desktop is sometimes not needed any longer to access the Internet [36]. But this is not the only thing helping mobile Internet to grow. Broader coverage of faster networks has also been an improvement factor.

2.1.6 Screen Form-Factors

The term screen form-factor describes the aspect ratio of the device's rendering surface width over its height. As the mobile sector grew the issue of representing a content, either web or native content, on mobile devices with a various number of different screen form-factors arose. When thinking of cross platform development, and therefore not only on mobile platforms, the fact that nearly every screen form-factor[16] can appear. So the shown screen resolutions in figure 2.2 have only a representational character to clarify the importance of the possibility of a responsive designed application, that adapts to the screen or window it is running on [37]. More on the topic of responsive design can be found in section 2.3.

[16] screen form-factor, or in the case of a desktop application, window form-factor.

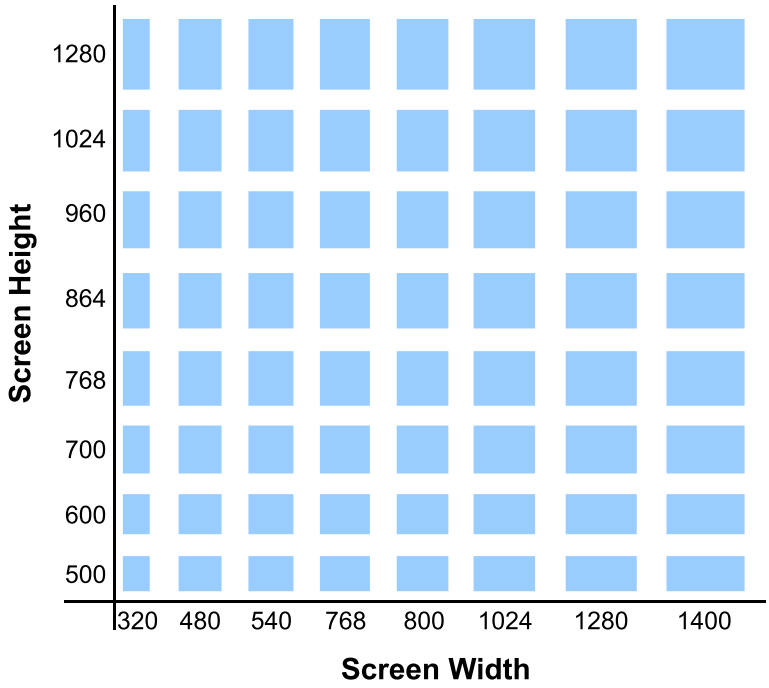

Figure 2.2: Overview of different possible screen form-factors and resolutions that can be found in mobile devices, as well as on desktop screens, in form of applications that run in window-mode with different form-factors.

2.2 Different Approaches to handle Screen Form-Factors

Within this section two different approaches are discussed; Desktop First and Mobile First As the name indicates the *Desktop First* approach is used to design the layout of a website for the desktop first and then adjust for a mobile version, whereas the *Mobile First* approach does it vice versa.

2.2.1 Desktop First

Before the appearance of smartphones and tablets most websites have been designed for desktops and laptops for years. The, now so called, Desktop First philosophy were used. Before the first iPhone and Android devices entered the market not a lot of people used the web on mobile devices, therefore the need for creating mobile websites or web applications was not given [36]. When taking the growth of the mobile sector, as mentioned in section 2.1, into consideration, starting with the design of a web product optimized for the desktop and then refactoring it for mobile devices may be an increasingly backward way of thinking [36]. For users of mobile devices it

is laborious to browse a desktop site in a mobile browser as is demonstrated on the right side of figure 2.3. Text can only be read when zooming in, which is uncomfortable and not user friendly. Some websites do not even work in a mobile browser because of the use of technologies a unsupported in mobile browsers [36].

2.2.2 Mobile First

Mobile First differs from Desktop First at the point that it takes care of the mobile of a website or web application first. Only techniques that work on all major mobile browsers can be used [36]. After the design of a mobile layout, a layout for desktops is created. When designing the mobile website first and then thinking about the desktop it can be ensured that the site provides a great user experience on both screen form-factors. The left side of figure 2.3 makes it more clear why responsive web applications are better to use on smartphones, than websites that are designed for desktop screens. The focus is on optimizing the reading experience and simplifying the navigation on devices with small screen form-factors [36]. In fact, some of the worlds biggest companies are suggesting the Mobile First philosophy. Google Chairman Eric Schmidt speaking at the DLD conference in Munich advises [62]:

> "The simple guideline is whatever you are doing – do mobile first."

Or Kevin Lynch Adobe's CTO [50]:

> "We really need to shift now to start thinking about building mobile first. This is an even bigger shift than the PC revolution."

With the revolution of capable mobile devices as shown in section 2.1.5 and the introduction of faster networks, mobile Internet usage has increased. Building a mobile version of the website or web application first not only takes advantage of this growth. Furthermore it gives the chance to provide an improved user experience for visitors and to reach more people [36], more often everyday because a user takes his smartphone with him most of the time [5].

2.3 Responsive Web Design (RWD)

To define the term and concept of RWD, this section deals with the definition and delimitation of the term responsive itself, as well as the origin of the name RWD, its main goals and the most common techniques: fluid grids, flexible images and Media Queries. It also shows ways to designing responsive are stated, followed by a short intro into common RWD frameworks and platforms.

Figure 2.3: Design of a website: Desktop version on a smartphone using only fluid grids to adapt to the screen-width (left); Responsive design using fluid grids and Media Queries to fit the smartphone (right) [66].

2.3.1 Definition and Delimitation

When defining RWD, the term responsive has to be clear. Therefore the definition of the word responsive itself is given after the definition of web design. The history of RWD and the delimitation to Responsive- versus Adaptive Web Design is stated to complement this section.

Responsive

According to the *Oxford Dictionaries* there are two definitions of the adjective *responsive* in English. The first definition – *reacting quickly and positively* e.g. *a flexible service that is responsive to changing social patterns* – can also be defined as – *responding readily and with interest* e.g. *our most enthusiastic and responsive students.* Examples for synonyms for responsive are *quick to react*, *reactive* and *receptive* [28].

A second definition of *responsive* is *in response; answering* e.g. *That was not a responsive answer to the question.* Of a section of liturgy using *responses* – e.g. *After that we will have some readings, and then the responsive liturgy which is in your newsletter inserts* [28].

When thinking of RWD the first definition of responsive stated in section 2.3.1 is suitable – *reacting quickly and positively* – as well as – *responding readily and with interest.*

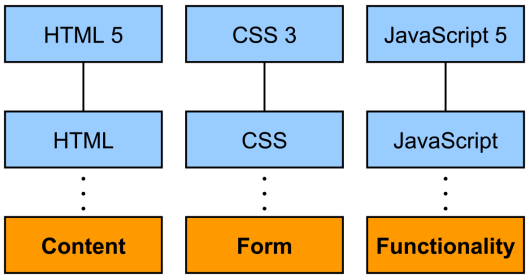

Figure 2.4: The subdivisions of content, form and functionality supplied to the three most relevant standards – HTML, CSS and JavaScript – in modern web design [3].

Web Design

When talking about web design in the context of RWD the first thing to consider is the distinction in development of websites in general between content, form and functionality. The content – text, images and other media contents – are described using HTML. The form – layout and graphical design – is specified in separate stylesheets such as CSS. To provide extended functionality and interactivity additional scripting languages are used. There is made a distinction between server-side scripting languages – e.g. PHP, Python, Perl, ASPNet, Cold Fusion or JSP – and widely client-side extensions – e.g. Flash, Silverlight, Java and JavaScript. Often the combination of HTML 5, CSS 3 and JavaScript is used in web design nowadays. Figure 2.4 clarifies the interaction of these three standards graphically. There are of course more than these three standards available when thinking of web design but in the context of RWD these are the most important ones [3].

HTML

A new standard of HTML (HTML 5 – status *Candidate Recommendation* in December 2012 [40]) was released as a cooperation between W3C[17] and WHATWG[18] [39]. HTML 5 was designed to replace HTML 4, XHTML[19] and the HTML DOM[20] Level 2. The intention was to deliver rich content without the need for additional plugins. Because HTML 5 is designed to run on a PC, tablets, smartphones or on a smart TV, it is also cross-platform. Some of the most interesting new features in HTML 5 are [41]:

- The `<canvas>` element for 2D drawing

[17] W3C: World Wide Web Consortium
[18] WHATWG: Web Hypertext Application Technology Working Group
[19] XHTML: Extensible HyperText Markup Language
[20] Document Object Model

- The `<video>` and `<audio>` elements for media playback
- Support for local storage
- New content-specific elements, like `<article>`, `<footer>`, `<header>`, `<nav>`, `<section>`
- New form controls, like calendar, date, time, email, url, search

Cascading Style Sheets (CSS) 3

The CSS 3 specification is still under development by W3C [27]. It has been split into *modules*. The *old CSS specification* are still included, but split into smaller pieces. Additionally new modules are added. Some of the most important CSS 3 modules are [26]:

- Selectors
- Box Model
- Backgrounds and Borders
- Image Values and Replaced Content
- Text Effects
- 2D/3D Transformations
- Animations
- Multiple Column Layout
- UI

JavaScript

To add functionality to websites, JavaScript is a common scripting-language nowadays. JavaScript can manipulate the HTML DOM to change HTML elements, attributes and styles (CSS), as well as validate data to validate user input [25]. JavaScript can e.g. be used to [45]:

- Change HTML elements
- Delete HTML elements
- Create new HTML elements
- Copy and clone HTML elements

History

The term RWD was coined by *Ethan Marcotte* in a May 2010 article in *A List Apart* [2]. He described the theory and practice of RWD in his 2011 published book titled *Responsive Web Design*. With the possibilities of RWD, he presents a way to escape static websites that deal with canvas with fixed boundaries or fixed-width layouts. He denounces that web design has to be independent from designing print media which he states that web design is derived from, to preserve the flexibility of different displays, screens

and browser windows. All the technologies Marcotte used to define RWD existed already: fluid grids, flexible images and Media Queries. Ethan Marcotte united these techniques under a single banner. With the following sentence Jeremy Keith summarize it quiet good [2].

> "Uniting these techniques he changed the way thinking about web design [...]."

The response of the media proved him right. In 2012 RWD was listed as *#2* in *Top Web Design Trends for 2012* by the *creative bloq* magazine [6]. Pete Cashmore stated the year 2013 *The Year of Responsive Web Design* [22]. And the trend, creating websites that respond on different screen form-factors, is still ongoing.

Responsive- versus Adaptive Web Design

RWD is not the only way to represent a website on mobile devices. There are techniques out there which are somehow similar to RWD, such as *Adaptive Web Design* (AWD). The term *Adaptive Web Design* was coined by *Aaron Gustafson* in his 2011 book with the same title [1]. The biggest similarity between the two methods is that they both allow web content to be viewed in mobile browsers and various screen form-factors. Both of them have the goal to provide visitors with a better mobile user experience. The way they differ is in their delivery of the structures. To summarize the differences with one simple sentence *Ryan Boudreaux* writes in his article *What is the difference between responsive vs. adaptive web design?* on *Web Designer* in April 11, 2013 [21]:

> "The distilled definition of a responsive web design is that it will fluidly change and respond to fit any screen or device size [...]" whereas "[...] adaptive design [...] will change to fit a predetermined set of screen and device sizes."

2.3.2 Main Goals

The objectives of RWD can be summarized with the following listings that *Matt Doyle* published in an 2011 article *Responsive Web Design Demystified* at *Elated* [29]:

- Adapting the layout (flexible) to suit different screen sizes
- Resizing images to fit the screen resolution
- Serving up lower-bandwidth images to mobile devices
- Simplifying page elements for mobile use
- Hiding non-essential elements on smaller screens

- Providing larger, finger-friendly links and buttons for mobile users, and
- Detecting and responding to mobile features such as geo-location and device orientation

2.3.3 Techniques

Within the next sections the three core techniques of RWD, defined by Ethan Marcotte, are explained:

- fluid grids,
- flexible images and
- Media Queries - a module from the CSS 3 specification.

With the combination of these techniques and rethinking about web design, it is possible to design more flexible, responsive websites.

Fluid Grids

As already stated in section 2.2, most websites were implemented with a fixed width style layout and centred content before fluid grids became popular. Nearly every computer display had the same screen form factor. Nowadays more and more different screen resolutions, aspect ratios and pixel densities are in use. Therefore a fixed width design is not a good solution, because it is not adaptable to different screen form-factors. Therefore liquid layouts became popular [66].

The main idea of fluid grids is to create a layout where all elements are based on relative width to other elements instead of a fixed pixel width. This leads to the effect that all elements in the layout are resizeable in relation to one another. To use this in practice it is necessary to stop thinking in pixel units and start thinking in proportions. For correct calculation of an element's proportion equation 2.1 is used,

$$r = \frac{t}{c} \tag{2.1}$$

where r is the result, relative on the target t divided by the context c the target is placed on. Moving the decimal over two places by multiplying the given value with 100, as shown in equation 2.2 gives a percentage value

$$20.83 = \frac{200 \cdot 100}{960} \tag{2.2}$$

that can be passed to the target element [66]. Giving the size of an element in percentage value a web designer can perfectly design the website for one resolution, and on changing resolutions the size of the elements within the website also changes. It is important to not only convert the sizes of

elements to a proportional value, but also the sizes of margins and paddings. Last but not least every pixel unit has to be replaced by percentage values.

One value that is indispensable to achieve a fluid grid design is the width of the browser window. When thinking of a fixed layout the term canvas is often used to delimit the width of the website. The width of the browser window serves the canvas with and is needed for further calculations of sizes of different elements. Access to the width of the browser window is needed during runtime to respond on windows that are resized e.g. on a desktop or mac [2].

Also *Dirk Jesse* in his article *Flexible Layouts: Challenge For The Future* at *Smashing Magazine* points out that the layout width has to be set in any percentage value or automatically using *widht:auto* to ensure the layout to use the whole browsers window width [46]. Accordingly to that he also suggests to set a minimal width in pixel value – to make sure that the content is accessible when displayed on a minimal screen resolution – and a maximal width in proportional value – thus the text flow does not grow in width in an uncontrollable fashion, but remains constant for various screen resolution [46].

Flexible Images

Using fluid grids also images have to be flexible. To set an image flexible its container has to be calculated with the formula for fluid grid elements, that is mentioned in section 2.3.3. When only calculating the containers size it might be that the image data bleed out of its containing element. Ethan Marcotte presents two possible solutions to make the images fluid dynamically scaling and cropping.

To scale the image dynamically to the size of its container the maximum width of the image is set to 100%. Setting the maximum width to 100% is a way to constraint the images to a manageable size. Alternately there is the possibility to simply clip off all that excess, overflowing data setting [2]. In section 3.4.3 of the implementation part of the thesis it is shown how an image looks like when it either gets scaled (figure 3.10) or cropped (figure 3.11).

Media Queries

Using Media Queries, a module of CSS 3, it is possible to adapt the content to conditions. A list of the features that can be tested in Media Queries is shown in table 2.1.

With that possibility websites can be designed to respond to different devices. The appearance of websites can be tailored to a specific range of output devices without changing the content itself, but only the style. For example the usage of multiple versions of one and the same image is possible

Table 2.1: Device features that can be tested in Media Queries [2] .

Feature Name	Definition	min/max prefixes
width	The width of the display area.	yes
height	The height of the display area.	yes
device-width	The width of the device's rendering surface.	yes
device-height	The height of the device's rendering surface.	yes
orientation	Accepts portrait or landscape values.	no
aspect ratio	Ratio of the display area's width over its height. E.g. on a desktop, you'd be able to query if the browser window is at a 16:9 aspect ratio.	yes
device aspect ratio	Ratio of the device's rendering surface width over its height. E.g. on a desktop, you'd be able to query if the screen is at a 16:9 aspect ratio.	yes
color	The number of bits per color component of the device. E.g., an 8-bit color device would successfully pass a query of (color: 8). Non-color devices should return a value of 0.	yes
color-index	The number of entries in the color lookup table of the output device. For example, media screen and (min-color-index: 256).	yes
monochrome	Similar to color, the monochrome feature lets us test the number of bits per pixel in a monochrome device.	yes
resolution	Tests the density of the pixels in the device, such as screen and (resolution:72ppi) or screen and (max-resolution:300ppi).	yes
scan	For tv-based browsing, measures whether the scanning process is either progressive or scan.	no
grid	Tests whether the device is a grid-based display, like feature phones with one .xed-width font. Can be expressed simply as (grid).	no

– knowing e.g. the width of an element and the resolution of the screen the designer specifies the image to display. With Media Queries also rearranging, scaling and grouping of the content is possible to make the website look more related to the device it is displayed on [2].

2.3.4 Designing Responsive

When starting to develop a website with a responsive characteristic one can e.g. write the HTML, CSS and JavaScript code, or use a framework. Typical components of RWD frameworks are

- grid system with multiple columns based on CSS,
- defined Typography for all HTML elements,
- browser compatibility and fallback-solutions for older browsers and various rendering engines (e.g. Mozilla, IE),
- standard CSS classes to built advanced classes,
- and JavaScript libraries e.g. jQuery for the animation of elements [34].

With the use of a framework the foundation for development and design of a website is built so that a developer does not have to start from the beginning every time. In general, using a framework results in advantages as well as disadvantages [34]. Advantages using a web design framework are

- the regular updates,
- the experiences and help in forums and communities,
- the best practices are often implemented,
- the reusability of source code for further websites,
- a flatter learning curve,
- and a faster mock-up processes [34].

Disadvantages using a web design framework are

- the dependency on the framework,
- the flatter learning curve,
- mixing of content and code – mostly [34].

A flatter learning curve is an advantage, since not everything has to be learned from the beginning itself, but also a disadvantage because the principles behind a framework are probably not understood by the developer.

Within frameworks and platforms, the Twitter Bootstrap, Designmodo's Startup Design Framework and Foundation 5 by ZURB are presented to give an overview of the key features of RWD frameworks.

2.3.5 Frameworks and Platforms

The following sections Twitter – Bootstrap, ZURB – Foundation, and Designmodo – Startup Design Framework deal with three specific frameworks

and platforms that can be used when developing a website with the RWD approach.

Twitter – Bootstrap

One of the most popular web design frameworks nowadays is the Bootstrap framework. The framework itself is free. It contains HTML and CSS-based design templates, as well as JavaScript extensions. Bootstrap was originaly developed by Mark Otto and Jacob Thornton at Twitter[21] for internal consistency across a wide range of tools before the framework was used for developing public usable interfaces. It got released on GitHub as an open source project in August 2011. In February 2012 it was one of the most starred GitHub development projects. In June 2014 it was the No.1 project on GitHub with 69,000+ stars and 25,000+ forks [20], [34].

ZURB – Foundation

Foundation emerged as a ZURB[22] project for faster and better development of front-end code. It is licensed under the MIT[23] License[24] as open-source and is available on Github[25]. Foundation contains HTML and CSS-based design templates, as well as optional JavaScript extensions. It is a free toolset for creating websites and web applications. Although Foundation has relatively incomplete support for HTML 5 and CSS 3 it is compatible with all major browsers. The actual release is Foundation 5.0 [33], [34].

Designmodo – Startup Design Framework

Designmodo provides a design framework and ready-made UI packages that can be used to develop a website quickly. It is especially for startup companies that need presence in the web. After purchasing the framework different UI packages can additionally be ordered. These packs are either free or also for sale. The company also provides a lot of informative material that can help designers and web developers according to topics such as Web Design and Web Development, Tips and Tutorials and WordPress [63], [34].

[21] Twitter Inc. is an online social networking and microblogging service that enables users to send so-called *tweets* – short 140-character text messages.

[22] ZURB is a privately held strategy and interaction design consulting firm. Major projects: designed sites including eBay, Facebook, Photobucket

[23] MIT: Massachusetts Institute of Technology

[24] MIT License is a free software license originated at the Massachusetts Institute of Technology [4].

[25] GitHub: a Git repository web-based hosting service.

2.4 Cross Platform Development

To immerse the field of developing cross platform applications, this section gives the definitions and delimitation. Within section 2.4.2 the development issues are given, followed by section 2.4.3 that describes the benefits and disadvantages when deciding to use a cross platform framework for development. To finish this section three selected frameworks with their key features are described.

2.4.1 Definitions and Delimitation

This section starts with a definition of the term platform. Furthermore a definition for cross platform is given. The declaration of platform independence is stated to delimit the term *cross platform*.

Platform

The term *platform* can refer to two general subgroups – *hardware platforms* and *software platforms* – or to *combination* thereof [49]. Hardware platforms themselves can either be a type of processor[26] or a hardware system[27]. Software platforms can refer to an operating system[28] or a programming language, such as Java. Java e.g. uses an operating system independent VM for its compiled code, the *Java Virtual Machine* (JVM)[29]. Java attempts to be cross platform by heaving a program's source code compiled into an intermediate bytecode language and then have it executed by the JVM that is written for the underlying hardware platform [44].

Cross Platform

Cross platform refers to the ability of software to operate on *more than one* platform with identical or nearly identical functionality. It differs from the term *platform independence*, which has a somehow similar meaning but implies that software will operate on *any* platform, whereas the implication of cross platform is that software will operate on at least two platforms [49].

Platform Independence

To give an example of platform independence, TCP/IP, the dominant networking protocol, can be mentioned. Therefore the Internet and web can

[26] Processortypes – e.g. ARM, x86, x86-64, PowerPC

[27] Heardware system – e.g. mainframe, workstation, desktop, handheld or embedded

[28] Operating system (OS) – e.g. Microsoft Windows, Mac OS X, Linux, Android

[29] Java executables do not run natively on the operating system, however the JVM is fully capable of providing os-related services (e.g. I/O, disk and network access.). If the appropriate privileges are granted. The protection level can be set by the user depending on an ACL.

also be considered as platform independent. This platform independence has been a major factor in the rapid growth of web-based applications, which are expected to provide increasing competition to operating system-based applications in the coming years [49]. Well designed web applications are platform independent because they can be executed from most modern browsers.

2.4.2 Development Issues

The main reason to develop cross platform is to provide the application to widespread user base, no matter which platform they run on. Designing software for more than one platform can be a time-consuming task because different operating systems have different APIs or run on different architectures. Nevertheless, there are two primary possibilities to develop cross platform. The first possibility is to compile to machine language. Therefore an executable program is compiled into the machine language and operating system of each target computer. This can be done using hardware-related languages e.g. C++ and creating separate sets of source code for each platform. The second possibility is to use an interpreter e.g. JVM [55]. To develop cross platform applications there are a number of frameworks, tool-kits and environments that can be used. Most dependent criteria to choose for one environment are the platforms that are supported and the language that is used for programming. Some developers prefer Java, others C++ and others again some scripting languages e.g. JavaScript or Lua. Within section 2.4.4, Adobe's PhoneGap, Corona Labs' Corona, and LibGDX by Mario Zechner are characterized in more detail.

2.4.3 Cross Platform Programming

Before starting to develop an application the developer has the choice to decide weather for cross platform programming or native application programming. When deciding for cross platform programming often cross platform frameworks are used instead of writing a framework for cross compiling from the very beginning. As with any approach, the use of cross platform frameworks results in advantages as well as disadvantages. Major advantages using a cross platform framework are

- the reusability of source code,
- the access to different plugins,
- and the reduction of development costs [65].

Disadvantages using a cross platform framework are

- the dependency on the framework,
- problems accessing features of a platform and
- a limited access to features of a platform [65].

The major advantage over native application development is that the code is reusable, and therefore reduces the costs in development, depending on the framework [65].

2.4.4 Frameworks

First the three Frameworks – Adobe PhoneGap, Corona SDK and LibGDX by Mario Zechner are introduced. Then these sections get expanded by a listing of the most relevant features, and a listing of the supported platforms.

Adobe – PhoneGap

The initial PhoneGap framework for mobile development was developed by Nitobi before being purchased by Adobe Systems in 2011 [7]. It enables developers to build applications for mobile Platforms using HTML 5, CSS 3 and JavaScript. Depending on the platform HTML, CSS and JavaScript code gets wrapped. Therefore the features of HTML and JavaScript are extended to work with the device. The resulting applications are hybrid, meaning that they are neither truly mobile native[30] nor purely web-based[31], but so called hybrid applications. Since version 1.9 it is possible to even mix native and hybrid code snippets. PhoneGap builds on Apache Cordova [11], an open source software [30].

Corona Labs Inc. – Corona SDK

Corona SDK has been created by Walter Luh, the founder of Corona Labs Inc.. Developers have the possibility to build mobile business applications and games for iOS and Android based devices. The company announced that the SDK will also support Windows Phone sooner or later. Integrated Lua is used as scripting language, layered on top of C++/OpenGL. With the switch to the newer graphic engine – Graphics 2.0, based on OpenGL 2.0 and shaders – cinematic effects – such as filters, generators and composites – can be applied to. There is a free version of the Corona SDK available for download. A basic developer licence can be purchased for 16€ per month, an extended *pro* version at the time of writing costs 49€ per month [24].

Mario Zechner – LibGDX Framework

LibGDX is a cross platform framework that is especially designed to create games. For programming, any JVM-compatible language can be chosen e.g.

[30] not native: layout rendering is done via web views instead of the platform's native UI framework

[31] not purely web-based: not just web applications, but are packaged as applications for distribution and have access to native device APIs

Table 2.2: Most relevant features of the three introduces frameworks [30], [24], [48].

	PhoneGap	*Corona SDK*	*LibGDX*
Accelerometer	x	x	x
Camera	x	x	x
Compass	x	x	x
Contacts	x	x	
File	x	x	x
Location	x	x	
Media	x	x	x
Network	x	x	x
Notification	x	x	
Storage	x	x	x

Java, Scala or Kotlin. The JVM bytecode then gets translated to the corresponding language that is needed – e.g. to JavaScript for a web application or to native ARM or x86 CPU[32] instructions to run on iOS using RoboVM for translation. It provides a unified API that works across all the platforms listed in table 2.3. Applications can be run and debugged on the desktop, natively, instead of deploying to any of the platforms it should run on after each change in code. When integrating third party libraries, leaderboards or multiplayer modes can be supported. The framework also comes with a variety of tools – e.g. the Gdx Setup UI, to keep the project setup simple. Section 3.3 describes why the LibGDX framework is chosen to develop the prototype application [48].

Features Overview

To summarize the features that can be accessed, using different cross platform frameworks, table 2.2 gives an overview. Because LibGDX has been created as a game development framework it has weaknesses regarding certain platform specific functions e.g. accessing the devices contacts, and location, and sending notifications, at the time the thesis was written.

Supported Platforms Overview

Table 2.3 gives an overview of the platforms that are supported by the specific cross platform frameworks. In contrast to the Corona SDK, which

[32]CPU: Central Processing Unit

Table 2.3: Supported Platforms of the three introduces frameworks [30], [24], [48].

	PhoneGap	*Corona SDK*	*LibGDX*
Android	x	x	x
Bada	x		
BlackBerry	x		x
Chrome			x
Firefox			x
Firefox OS	x		
iOS	x	x	x
IE			x
Java Applet			x
Linux			x
Mac OS X			x
Opera			x
Safari			x
Symbian	x		
Tizen	x		
Ubuntu Touch	x		
webOS	x		
Windows			x
Windows Phone	x	announced	

only supports the most common mobile platforms, PhoneGap also supports rather unconventional platforms. The LibGDX framework is the only one that has support for browsers and also desktop platforms.

2.5 Conclusion

In section 2.1.1 it has been demonstrated, that there were separate devices needed, before the functionality of mobile phones were combined with several other functionality, e.g. camera, GPS and mobile Internet. The revolution in the mobile sector happened very fast. Section 2.1.2 explains that key players of early mobile operating systems have been largely replaced by Google with Android and Apple with iOS starting in 2007, with the release of the first iPhone. Beside Android and iOS there are e.g. Microsoft's Windows Phone

and the BlackBerry 10 operating system and still several other rather exotic OS e.g. Firefox OS. The shipment of mobile devices increased dramatically for the period 2008 to 2012, as stated in section 2.1.5, predicting that the trend continues to 2015. The portfolio of the various devices with different operating systems or different screen form-factors, read in section 2.1.6, that are out there nowadays will probably be expanded in the future. Due to the increase of mobile device sales and the extension of the mobile web, usage of mobile web applications increases. This leads to need of mobile websites. With the rethinking in modern web design, the Mobile First approach obsoletes the Desktop First approach. Different techniques are used to present the content of websites in a proper way. One technique is RWD. RWD refers to the use of already existing technologies, united under the term Responsive Web Design by Ethan Marcotte in 2010, to create a fluid layout, that responds on the screen form-factor it is displayed on. Core technologies are fluid grids, flexible images, and Media Queries, more accurate explained in section 2.3.3. RWD is currently used in web design. Sometimes native applications are preferred over web applications. In native application development cross platform frameworks enable the possibility to reach more than one platform. The terms platform, cross platform, and platform independence are described in section 2.4.1. It depends on the developer to either develop the application native for every platform the application has to run on, or selecting a cross platform framework, to write the code only once and let the framework compile it accordingly the needed platforms. The possibility of adopting principles of RWD, and making use of them in cross platform development forms the transition to the next chapter.

Chapter 3

Practical Approach

First part of this chapter explains the purpose of the sample application, that is explained in section 3.1. Due to the fact that the prototype application is implemented in cooperation with APEX gaming technology GmbH, that cooperation is described in section 3.2. Section 3.3 validates the selection of the cross platform framework, followed by section 3.4 that deals with the implementation of the most challenging parts of the prototype application. Section 3.5 finally summarizes the whole chapter.

3.1 Purpose of the Sample Application

The purpose of the prototype application is to prove whether it is possible to develop an application that shows a responsive behaviour within the environment of a specific cross platform framework and programming language, or not. Therefore the problem statement is stated in section 3.1.1. Section 3.1.2 is responsible for specification and delimitation of the actual prototype application.

3.1.1 Problem Statement

It is assumed that the application should run on the most common mobile and desktop platforms and be implemented using a cross platform framework. There are many facts to be considered:

- For what screen sizes should the layout be done?
- Should the application respond to different screens?
- Is more than one layout needed for different screen form factors?
- What are the minimum/maximum screen sizes the application runs?
- Are UI elements accessible easily and intuitive?
- Is readable content legible on different screen form factors?

How can it be guaranteed that the legibility, usability and accessibility is given on devices with different screen properties? The satisfaction of these constraints can only be measured with massive user studies which is not part of this thesis. Only the redemption of the hardware constraints – specifications of the needed device features and device delimitations – described in section 3.1.2 can be confirmed. These constraints are needed to delimit the amount of devices to an acceptable number that can also be tested.

3.1.2 Specification and Delimitation

Confirming weather it is possible to develop a cross platform application that represent itself in a responsive way or not a prototype application is written. The application has to deal with different screen form factors. For the purpose of this thesis the devices are delimited with the iPhone 3GS – 3.5" display, an aspect ratio of 3:2, a resolution of 480 × 320px, and the orientation in landscape mode – at the minimum, with the following hardware specifications – , and a desktop screen – 24" display, an aspect ratio of 16:9, a resolution of 1920 × 1200px, and the orientation in landscape mode – at the maximum. To confirm the cross platform approach, the application has to run on at least two platforms, where at least one is a mobile platform. The techniques described in section 2.3.3 have to be applied to the application. Table 3.1 is a set of device features that has to be tested within the application. Most of the features are taken from table 2.1 of section 2.3.3, only the last feature is added to provide a more platform specific style to the layout management.

3.2 Cooperation with APEX gaming technology GmbH

As the main parts of the prototype application for this thesis were developed in a cooperation with APEX gaming technology GmbH this section gives a short overview of APEX gaming technology GmbH. Therefore it is split up into section 3.2.1 that gives an insight into the company itself, and section 3.2.2 that describes the reason why it has been decided to go responsive at APEX.

3.2.1 About the Company

APEX gaming technology GmbH is an international manufacturer and operator in the gaming industry, from producing slot-machines and roulette wheels to mobile gaming platforms. It is a privately-owned company, founded and managed by Mr. Johannes Weissengruber. The global headquarters of APEX gaming are based in Hagenberg im Mühlkreis, Austria. Back in 2003

Table 3.1: Device features that has to be tested within the application to provide a responsive characteristic.

Feature Name	Definition	min/max prefixes
width	The width of the display area.	yes
height	The height of the display area.	yes
orientation	Accepts portrait or landscape values.	no
aspect-ratio	Ratio of the display area's width over its height. E.g. on a desktop, you'd be able to query if the browser window is at a 16:9 aspect ratio.	yes
resolution	Tests the density of the pixels in the device, such as screen and (resolution:72ppi) or screen and (max-resolution:300ppi).	yes
style guideline	Adapt the style of the application according to platform specific guidelines (e.g. min-max sizes of buttons/paddings).	no

Mr. Weissengruber chose the company name to reflect his strategy. The Oxford English Dictionary explains the meaning of the word *apex* as *peak, top or highest part* – which in this case suits for the highest quality possible in every aspect. The company logo of the pyramid with the golden shaft on top – the apex – underlines this strategy. It was back in 1994 that Mr. Weissengruber first entered into the gaming industry. When the opportunity arose to take a 100% stake in this operating company called *Play and Win* a year later, he did not waiver. It is his true fascination of how the industry works and – more importantly – how players accept and play on slots games – that are the roots of APEX success. Alongside the headquarters in Austria and the manufacturing base in the Czech Republic, APEX gaming runs wholly-owned subsidiaries in Albania, Austria, Czech Republic, Germany, Macedonia, Mexico, Serbia and Spain. In addition to this, APEX gaming is present in many more countries, working together with local partners [12].

3.2.2 Going Responsive at APEX

The current APEX Slot Challenge application is implemented in Corona SDK using Lua as programming language. Due to several problems using the Corona SDK, also described in section 3.3, the decision was made to develop a prototype application in LibGDX using Java as programming language. The reason that one of the non-mobile slot machines of APEX bases on Java,

facilitated the decision which cross platform framework to chose. More about the selection of the LibGDX framework can be read in section 3.3. When starting from scratch several decisions were made and weighted according to their importance. One decision that is made and rated a high importance is the responsive characteristic that the prototype application has to show to adapt to the different screen form-factors of nowadays mobile devices, described in section 2.1.6.

3.3 Selection of the Cross Platform Framework

For the implementation of a prototype application the LibGDX framework has been chosen from the list of the three cross platform frameworks, characterized in section 2.4.4. The decision was made on the following aspects in order of their importance

- native application,
- programming/scripting language(s),
- and possible platforms.

To use the best practices of RWD accurately, Adobe's Phone Gap framework might have been the right choice. Due to the fact that the source is a website – written in HTML in combination with CSS and JavaScript – that is ported to mobile platforms as a hybrid application it would have been easy to establish an application with a responsive characteristic. The fact that the application is hybrid is the problem. When thinking of a business application that is heavily based on text and some photos that gets updated from time to time, Phone Gap would have been the right choice. When thinking of APEX Slot Challenge, a heavily customized gaming application that changes content in a very fast way – for example spinning of reels – and also has some time-critical parts that has to be obeyed – Phone Gap was eliminated as a possible framework.

Corona SDK as the next framework on the list is already in use at the mobile development department of APEX. Due to the disadvantages regarding debugging, handling of graphical content and textual representations, that the Corona SDK – when the choice for a framework was made – the Corona SDK was eliminated as possible framework.

The reason why the choice fell on the LibGDX framework was that it is possible to write applications in Java with all the benefits of the *Java-Compiler*[1] e.g. better refactoring, static type checking, powerful IDEs[2]. Additionally the application can be cross-compiled natively for more platforms than the other two frameworks, visible in table 2.3.

[1] Java-Compiler: part of the development tool that translates the source-code in the so-called Java-Bytecode, that is directly executable by the JVM

[2] IDE: Integrated Development Environment – e.g. Eclipse

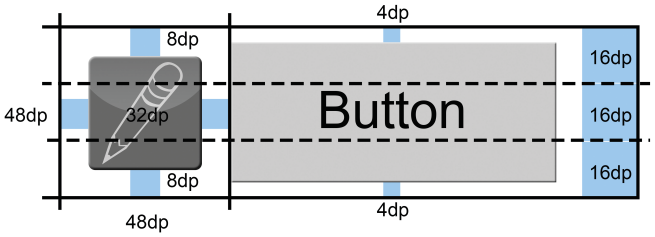

Figure 3.1: Metric's according to the android design guidelines [8].

3.4 Actual Application

That section leads to the implementation of the most interesting parts of the actual application. To get to that the most necessary mathematical notations are stated, followed by the application's architecture. Finally the section finishes with the implementation in section 3.4.3. The focus of the implementation is on the realization of the responsive characteristic of the application, and therefore only the most interesting parts of this field are stated and described in detail.

3.4.1 Mathematical Notations

When looking at the table 3.1 there are features that are needed to prepare for fluid grids. The width and height of the displays area is the delimitation for the application. When working with LibGDX these features are represented in pixel units. The mathematical equation 2.1 to realize fluid grids. A target can be any UI element – actor in LibGDX – e.g. a simple button. When going through some platform specific UI guidelines a minimal button size is proposed. According to the Android design guideline the minimum button size is 48 dp – shown in figure 3.1 [8].

Device independent pixels (dp)[3] is used to ensure the same physical size of elements no matter on what device they are displayed on. It is Androids advice to describe an elements size information (for UI elements as well as padding/margin) in dp [8]. But there is also diversity to what dp is. On an Android device one dp is equivalent to one physical *pixel* (px) on a screen with 160ppi. Whereas the Windows Presentation Foundation specifies one dp as equivalent to $1/96^{\text{th}}$ of an inch [61]. For the purpose of this thesis, Android's approach of dp is taken. To explain Android's way of calculating dp, the equation 3.1 is given. The result of the equation a is the amount of dp that is needed to express an object of a specified size b in px on a screen with a given resolution of $c$ppi.

[3] device independent pixel also: density-independent pixel.

$$a = b \cdot \frac{160}{c} \tag{3.1}$$

With the reverse statement shown in equation 3.2 the result b in px expresses the amount of px that is needed for a given size a in dp on a specific screen with a given resolution of $c$ppi.

$$b = \frac{a \cdot c}{160} \tag{3.2}$$

In contrast to websites most applications are restricted to the size of the screen they are shown on, which is another point that has to be considered. Equation 3.3

$$d_c = \sum_{i=1}^{n} d_i + \sum_{i=1}^{n+1} p_i + \sum_{i=1}^{n+1} m_i \tag{3.3}$$

gives the dimension d_c[4] of the context that is needed to place a specified number of n targets on it, with given dimensions d_i[4], padding p_i and margin m_i of the i^{th} target. That equation is used twice to calculate if all the targets fit horizontal and vertical. If the outcome of that equation is bigger than the given screen dimension, either vertical or horizontal, the targets do not fit the given context. One possibility to solve this problem is the use of Media Queries, described in section 2.3.3.

3.4.2 Architecture

Within this section the 2D scene graph basics of LibGDX are scribed to give a short insight into its scope of functions. Furthermore the application design of the prototype application is presented, in section 3.4.2, and also the system architecture of the prototype application, in section 3.4.2.

2D Scene Graph Basics of LibGDX

The class *Stage* of the *scene2d* package of the LibGDX framework is responsible to handle the viewport and distribute input events. A Stage contains hierarchies of the class *Actor*. An Actor is a 2D scene graph node and has a position, rectangular size, origin, scale, rotation, Z index, and color. The functionality of an Actor is applied by *Actions*. *Group* and *Widget* directly extend from Actor. Further relationships can be taken from figure 3.2. Any *Group* may contain other instances of Actor. *Widget* and *WidgetGroup* provide a minimum, preferred, and maximum size that is almost always overridden by a subclass [48].

[4]The dimension can either be width or height.

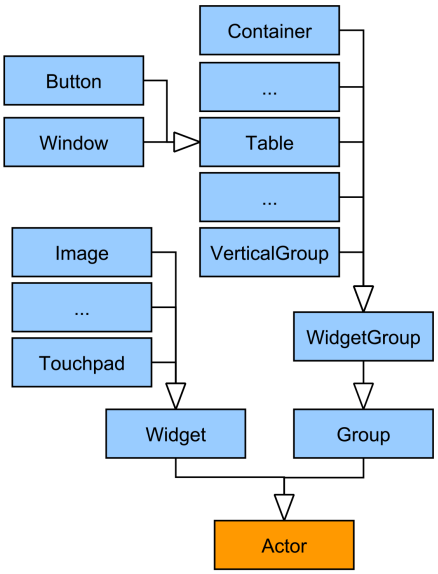

Figure 3.2: Part of the 2D scene graph to reveal the relationship between *Image*, *Table* and *Button* with the Actor class.

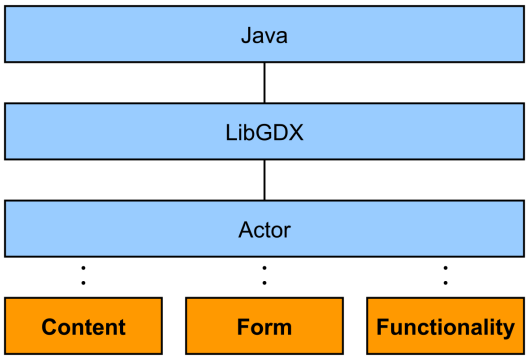

Figure 3.3: With the use of LibGDX Actor, there is no subdivision in Content, Form, and Functionality as in RWD.

Application Design

Figure 3.3 reveals the possibility that is chosen to design an application with a responsive characteristic in the LibGDX framework. The differences to figure 2.4 from section 2.3.1 get clear very fast. With the use of Java and LibGDX the separation of *Content*, *Form* and *Functionality* is united by the use of Actors.

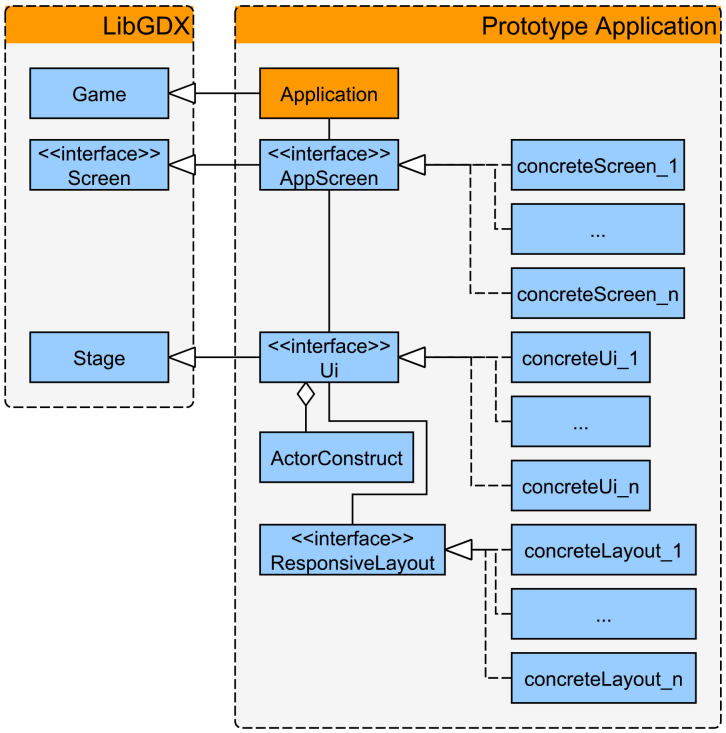

Figure 3.4: System architecture of the prototype application.

System Architecture

The class diagram in figure 3.4 shows the structure of the most relevant classes within the application that has been developed and how they are related to each other. *Application* – the core class of the prototype – shows an instance of the interface *AppScreen* according to the current state of the application. Every AppScreen has one or more instances of the interface *Ui* – e.g. an instance that represents a main menu of the AppScreen and another one for a drop-down menu. The Ui consists of a collection of *ActorConstructs* – that holds the Actor and other informations like an additional id and type – and an instance of the interface *ResponsiveLayout* – that is applied accordingly to the platform the application runs on.

3.4.3 Implementation

In the following section the implementation of the most relevant principles adopted from RWD are declared; fluid grids, flexible images, and Media Queries. The example implementations rely mostly on each other. The source code used in this sections provides a proof of concept and does not

by any means reflect best practices in software engineering. Premise for the implementation is a new Stage object *stage* to display and interact with the actors. The reference design for the desktop platform from figure 3.5 is made for a screen with a resolution of 960 × 720px which is a screen aspect ratio of 4:3. To show what happens when the application runs on a desktop screen in window-mode, with a different aspect ratio of 3:2, it is displayed on a resolution of 600 × 400px. Android is taken as the reference for the mobile platform. The reference design for Android, which can be seen in figure 3.6, is designed for a device with a comparatively low resolution of 480 × 320px, which is an aspect ratio of 3:2, and corresponds to the screen of the iPhone 3GS. The implementation of three key techniques of RWD, using the LibGDX framework with Java as programming language, can be found on the attached DVD.

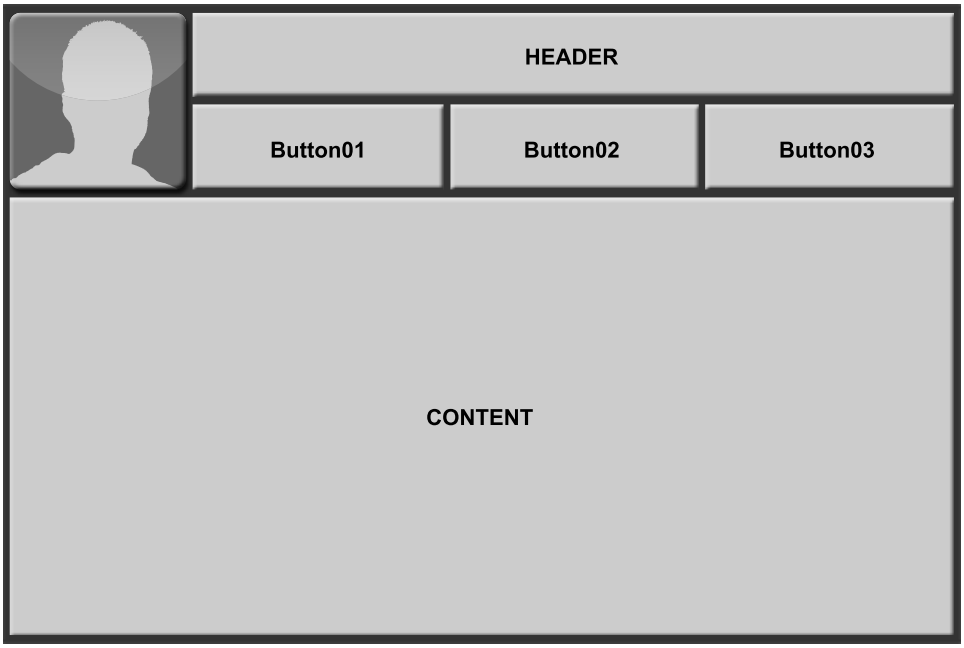

Figure 3.5: Initial design for the prototype application on a desktop screen. Window resolution: 960 × 720px.

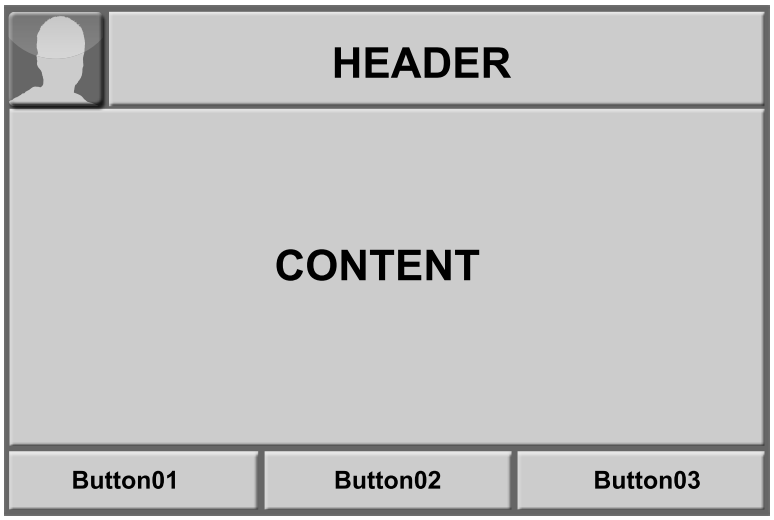

Figure 3.6: Initial design for the prototype application on a desktop screen. Display resolution: 480 × 320px.

Fluid Grids

Within this section of the implementation the principle of fluid grids is proven. To start with the implementation of fluid grids some actors are needed. First of all a new TextButton *button* is created and added to the stage, shown in listing 3.1. The positioning of the button is in the center of the stage. A peculiarity of LibGDX is that setting the origin of an object does only affect rotation and scaling, but not the translation. To set the button's position center of the stage, half of the button's width and height have to be subtracted. Because there are no other parameters specified the button stays at it's minimum size to represent the text in it's nested label.

Listing 3.1: Creation of a standard TextButton in LibGDX.

```
1 ...
2 TextButton button = new TextButton("Standard TextButton",
      getTextButtonStyle());
3
4 button.setOrigin(b_width / 2, b_height / 2);
5 button.setPosition(currentWidth / 2 - b_width / 2, currentHeight / 2 -
      b_height / 2);
6
7 stage.addActor(button);
8 ...
```

According to the design for the desktop application shown in figure 3.5 the element in the top left corner has a squared size of 200 × 200px. With the equation from section 3.4.1 the relative size of actors and margins are

Figure 3.7: TextButton that is designed to be squared – 200 × 200px on a 960 × 720px screen – not quadratic anymore due to change of aspect ratio of the displayed screen.

expressed mathematically. The implementation is shown in listing 3.2.

Listing 3.2: Make the TextButton relatively to the width of the screen.

```
1 ...
2 TextButton button = new TextButton("...", getTextButtonStyle());
3
4 float b_width = 200.0f / designedWidth * currentWidth;
5 float b_height = 200.0f / designedHeight * currentHeight;
6 float m_width = 20.0f / designedWidth * currentWidth;
7 float m_height = 20.0f / designedHeight * currentWidth;
8
9 button.setSize(b_width, b_height);
10 button.setText("...");
11 button.setOrigin(..., ...);
12 button.setPosition(m_width, currentHeight - b_height - m_height);
13
14 stage.addActor(button);
15 ...
```

The label of the button in figure 3.7, that is fed with its current size, shows exactly the expected result, but not what is wanted. According to the design the button has to be squared. Unlike in most web applications a native application often does not have endless scroll, but usually displays

all the needed UI elements on the screen. Therefore the UI elements depend not only on the width, but also on the height of the screen. In figure 3.7 the button is not squared anymore due to a change of the aspect ratio from 4:3 to 3:2 of the screen the application is displayed on. An easy way to keep the button within its required aspect ratio is to use the smaller value for width and height and multiply it with the needed aspect ratio. In this example the button is quadratic so the multiplication is not needed. The implementation in listing 3.3 applies the minimum size to the actor and the padding.

Listing 3.3: Make the TextButton relatively not only to the width of the screen, but also to the screens height.

```
1 ...
2 TextButton button = new TextButton("...", getTextButtonStyle());
3
4 float b_width = 200.0f / designedWidth * currentWidth;
5 float b_height = 200.0f / designedHeight * currentHeight;
6 float m_width = 20.0f / designedWidth * currentWidth;
7 float m_height = 20.0f / designedHeight * currentWidth;
8 float b_size = Math.min(b_width, b_height);
9 float m_size = Math.min(m_width, m_height);
10
11 button.setSize(b_size, b_size);
12 button.setText("...");
13 button.setOrigin(..., ...);
14 button.setPosition(m_size, currentHeight - b_size - m_size);
15
16 stage.addActor(button);
17 ...
```

Now that can be a problem for big displays. With this approach the margin is relatively larger on big displays than on small ones. This might be wanted in some cases, but in this case the use of a margin in dp units – introduced in section 3.4.1 – is appropriate. With the use of dp it can be ensured that the margin always stays the same on every screen and is not dependent on the screens size, resolution and pixel-density. In the following listing 3.4 the margin is set to 8dp and a Lable *lable* is added to the left side of the button. The label uses the remaining width of the window, taking the margins into account. The result of this implementation can be seen in figure 3.8.

Listing 3.4: Use dp instead of relative sizes for margins/paddings to ensure a consistent distance between the individual UI elements.

```
1 ...
2 TextButton button = new TextButton("...", getTextButtonStyle());
3 Label label = new Label("...", getLabelStyle());
4
5 float b_width = 200.0f / designedWidth * currentWidth;
6 float b_height = 200.0f / designedHeight * currentHeight;
7 float b_size = Math.min(b_width, b_height);
```

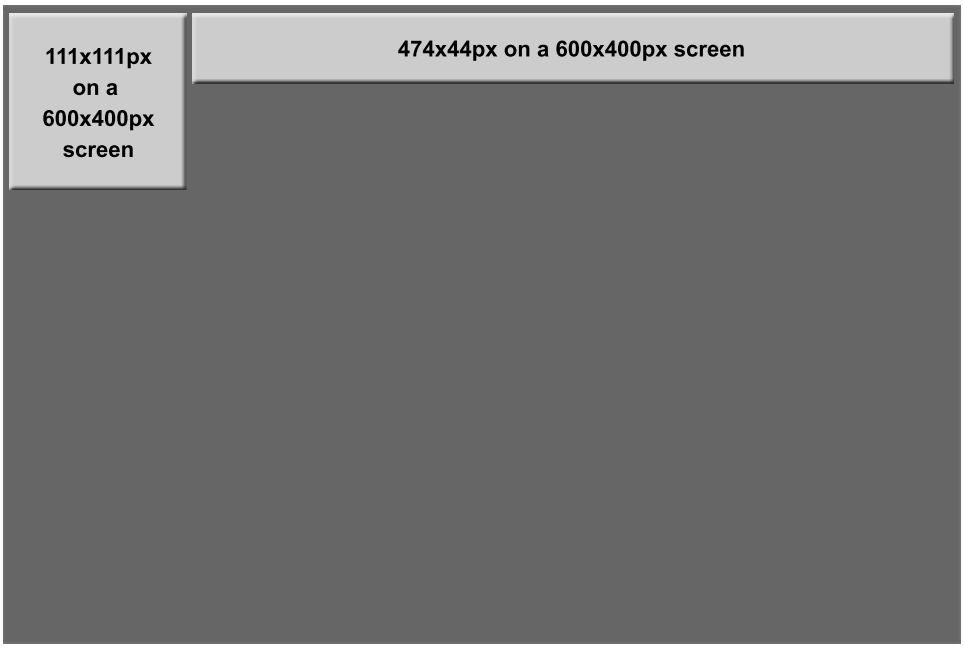

Figure 3.8: TextButton that is designed to be squared – 200 × 200px on a 960×720px screen. Would have been 125×125px on a screen with a resolution of 600 × 450px but the screen it is displayed on has only 600 × 400px. To stay squared the button takes its minimum value for width and height – 111 × 111px. The Label uses the remaining width of the window. Expressed margin in dp to stay the same independent of any screen parameters.

```
8  // Using 8dp
9  float m_size = dpInPx(8);
10
11 // |margin|button|margin|label|margin|
12 float l_width = currentWidth - b_size - 3 * m_size;
13 float l_height = 80.0f / designedHeight * currentHeight;
14
15 button.setSize(b_size, b_size);
16 button.setText("...");
17 button.setOrigin(..., ...);
18 button.setPosition(m_size, currentHeight - b_size - m_size);
19
20 label.setSize(l_width, l_height);
21 label.setText("...");
22 label.setOrigin(..., ...);
23 label.setAlignment(Align.center);
24 label.setPosition(2 * m_size + b_size, currentHeight - l_height - m_size
       );
25
26 ...
```

The technique stays the same for every actor that is added to the stage. First the values for the actor are set, it get initialized according to its dependencies on other actors or the screen. Afterwards it is setup and add to the stage. The listing below again shows exactly the steps when setting up an actor using the fluid grid approach. In figure 3.9 that approach of fluid grids is used to complete the design from figure 3.5 with all the needed UI elements.

1. Set values for actors.
2. Initialize actors according to their dependencies on each other.
3. Setup actors.
4. Add actors to stage.

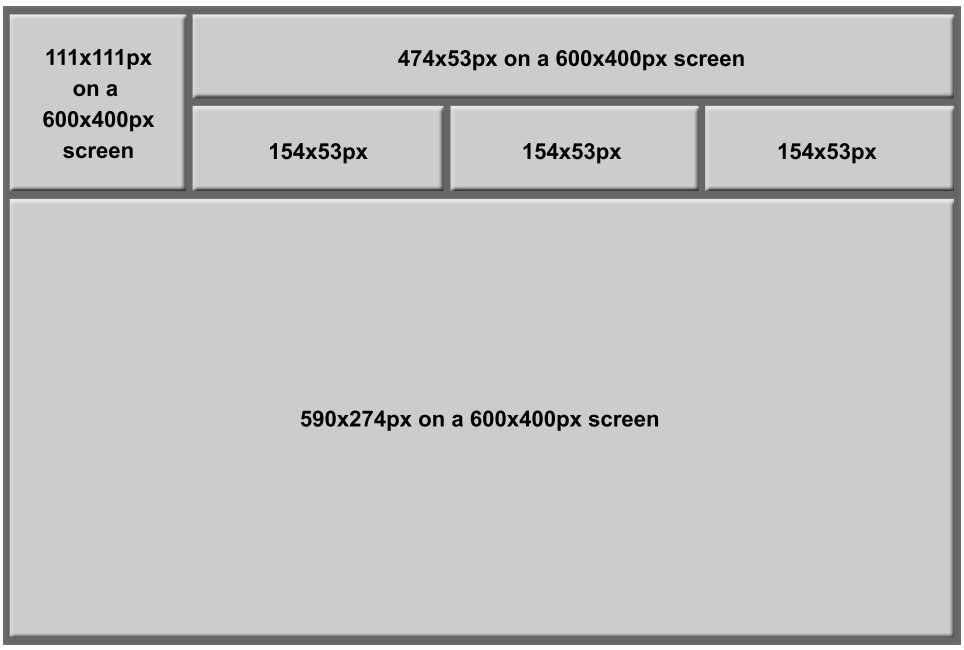

Figure 3.9: Implementation of the reference design for desktop.

Flexible Images

As mentioned before in the flexible images part of section 2.3.3 there are two possibilities for flexible images; Dynamic scaling and cropping. Within the code section beyond the TextButton from the top left is replaced by an Image *user* that represents the avatar of the user for the actual application. In LibGDX an Image can have a Texture. By setting the Texture and placing adding the image of the avatar to the screen, the texture gets scaled dynamically by default. An improvement would be to use different stages of resolutions for the image and let the program chose which one fits best.

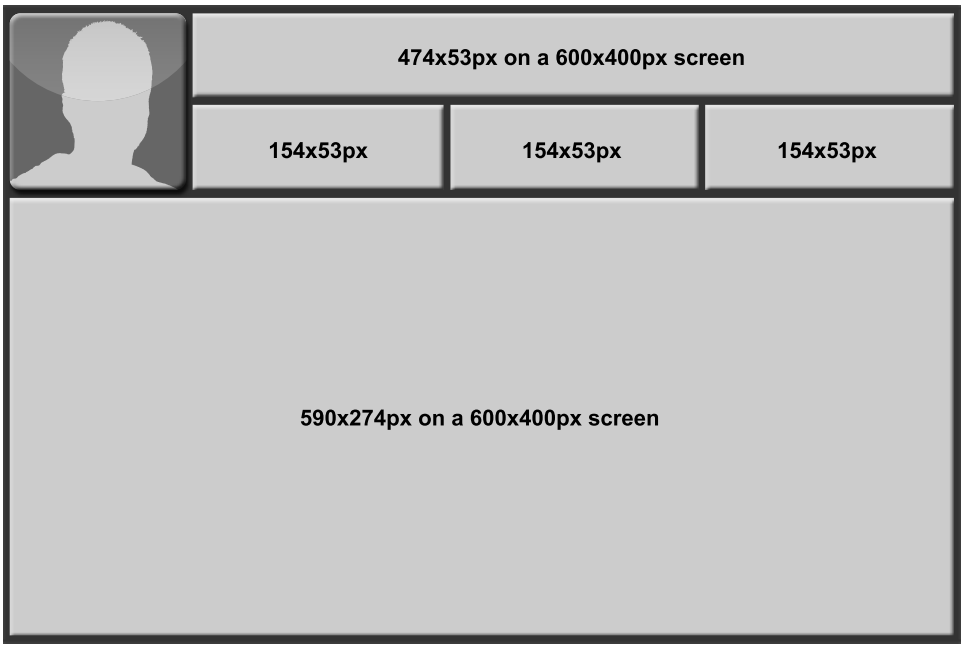

Figure 3.10: Dynamically scaled Image instead of TextButton on the top left corner to represent the avatar of a user.

That part is skipped for the implementation of the prototype application developed as proof of concept for this thesis. The outcome of this implementation in listing 3.5 can be seen in figure 3.10.

Listing 3.5: Initializing an image by using a Texture object for dynamical scaling.

```
1 ...
2 Image user;
3 ...
4 // Values for actors
5 ...
6 // Initialize actors according to their dependencies on each other
7 ...
8 Texture userTexture = new Texture(..."defaultImage.png");
9 user = new Image(userTexture);
10 user.setName("UserImage");
11 ...
12 // Setup actors
13 ...
14 user.setSize(b_userSize, b_userSize);
15 ...
16 // Add actors to stage
17 ...
```

LibGDX provides also a possibility for cropping, another possibility to make an image flexible. To crop an image a TextureRegion can be used. By referring to only a region of a specified texture, the given element gets cropped automatically. When having a look at the source code in listing 3.6 the size of the image is set directly when initializing the TextureRegion. Furthermore a position can be set when initializing the TextureRegion to get another part of the texture. In figure 3.11 the image representing the users avatar is cropped.

Listing 3.6: Using a TextureRegion for cropping of the image at a specified size.

```
1 ...
2 Image user;
3 ...
4 // Values for actors
5 ...
6 // Initialize actors according to their dependencies on each other
7 ...
8 Texture userTexture = new Texture(..."defaultImage.png");
9 TextureRegion userTextureRegion = new TextureRegion(userTexture, (int)
      b_userSize, (int) b_userSize);
10 user = new Image(userTextureRegion);
11 user.setName("UserImage");
12 ...
13 // Setup actors
14 ...
15 // Add actors to stage
16 ...
```

Media Queries

With the access to the device features specified in section 3.1.2 the principle of Media Queries from RWD can be adopted. An example for the use of Media Queries is that the design exceeds the screen limitations, e.g. there are too many UI elements placed on the stage. The calculation therefore can be found in section 3.4.1. With the use of Media Queries such problem can be eliminated. To check if all the actors fit the screen a helper method is written. In listing 3.7 a helper method is shown that displays a WARNING on the screen. It compares the initial size, that is set in LibGDX for an actor, with the size that is calculated for that actor. If the initial size is bigger than the calculated size the warning is printed on the application screen, visible in figure 3.12. Now the developer has the opportunity to redesign the current screen to e.g. fit smaller displays, or to exclude devices that are too small to display that application. When deciding for the first option there are several possibilities for the developer to continue. One is to group actors that are similar or have similarities into widgets e.g. a drop-down menu instead of displaying all actors on the screen. Another option, especially used

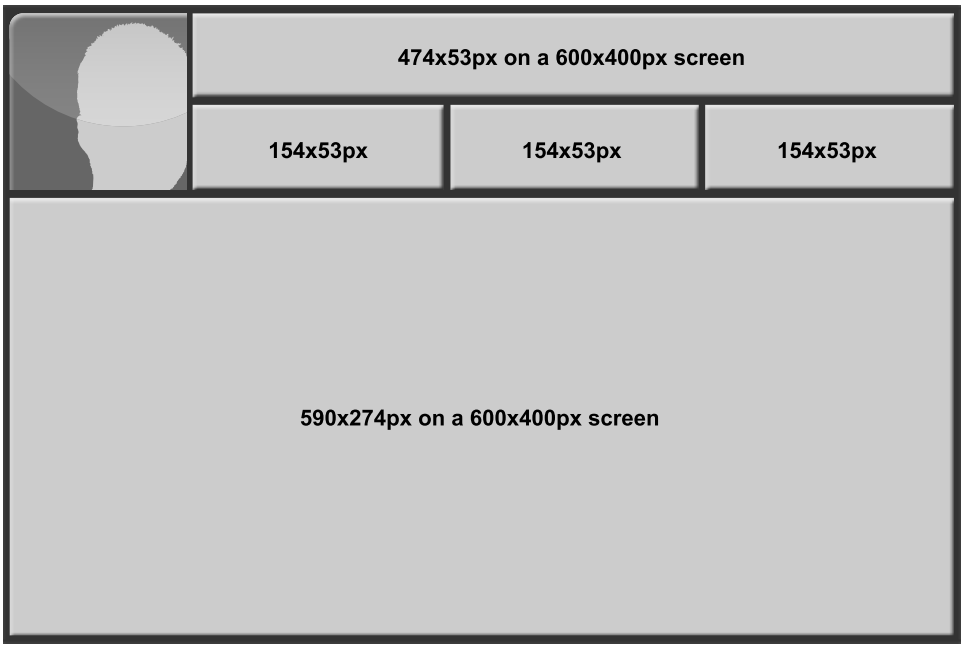

Figure 3.11: Cropped Image to show another possibility of a flexible use of images.

to display large text on screen, is a *ScrollPane*, to enable scrolling.

Listing 3.7: Displaying a WARNING message when the display gets too small for the UI elements that gets displayed.

```
1 ...
2 // Values for actors
3 ...
4 // Initialize actors according to their dependencies on each other
5 ...
6 // Check fitsStage of actors
7 boolean b_fitsStage = actorStageCheck(button, b_size, b_initialSize, "
      size");
8 if (b_fitsStage) {
9   button.setSize(b_size, b_size);
10 }
11 ...
12 // Setup actors
13 ...
14 // Add actors to stage
15 ...
16
17 private boolean actorStageCheck(actor, size, initialSize, ...) {
18   if (initialSize > size) {
19     float diff_size = size - initialSize;
20     String errorMsg = "...";
```

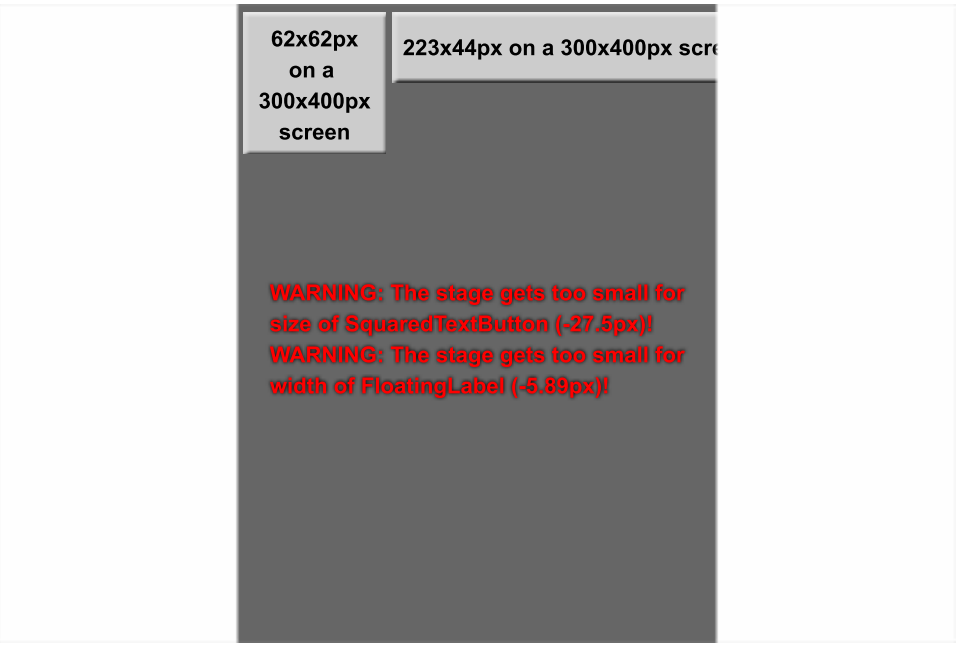

Figure 3.12: Error message shown on display when stage gets too small to display the actors on it without cropping them.

```
21      Gdx.app.debug("WARNING", errorMsg);
22      ...
23      return false;
24    }
25    return true;
26  }
27
28  ...
```

Another part that can be checked is the platform the application is running on to use values corresponding to platform specific design guidelines if available. A simple way to do this is the switch statement shown in listing 3.8. It switches according to the platform the application is displayed on. Assuming that mobile device screens are in most cases smaller than desktop screens a different design is applied to better exploit the space on mobile devices.

Listing 3.8: Applying different designs to the prototype application, according to the platform the application is running on, using a simple switch statement.

```
1 ...
2 // Values for actors
3 ...
4 // Initialize actors according to their dependencies on each other, check fitsStage
```

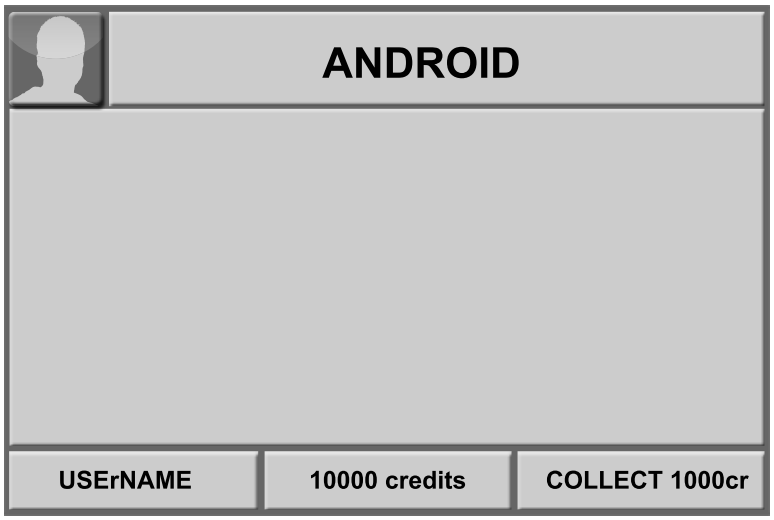

Figure 3.13: The prototype running on Android with the resolution of the iPhone 3GS (480 × 320px).

```
5  // and setup actors according to the platform the application runs on
6  ...
7  switch (PLATFORM) {
8      case Android:
9      ...
10     break;
11     case Desktop:
12     ...
13     break
14     ...
15 }
16 ...
17 // Add actors to stage
18 ...
```

The figure 3.13 represents the application on the minimum screen resolution corresponding the screen of the iPhone 3GS, realizing the design from figure 3.6. Figure 3.14 represent the application on a desktop screen with a resolution of 1920 × 1200px. This figure shows the implementation of the different look for the desktop application specified in the design of figure 3.5. With figures 3.13 and 3.14 it is proved that the prototype application is running on the minimum and maximum screen form-factors defined in section 3.1.2 and also on at least two different platforms, Android and Desktop.

3.5 Conclusion

The given hypothesis from section 1.3 lays the foundation of this thesis. The purpose of the sample application is to prove whether the hypothesis is

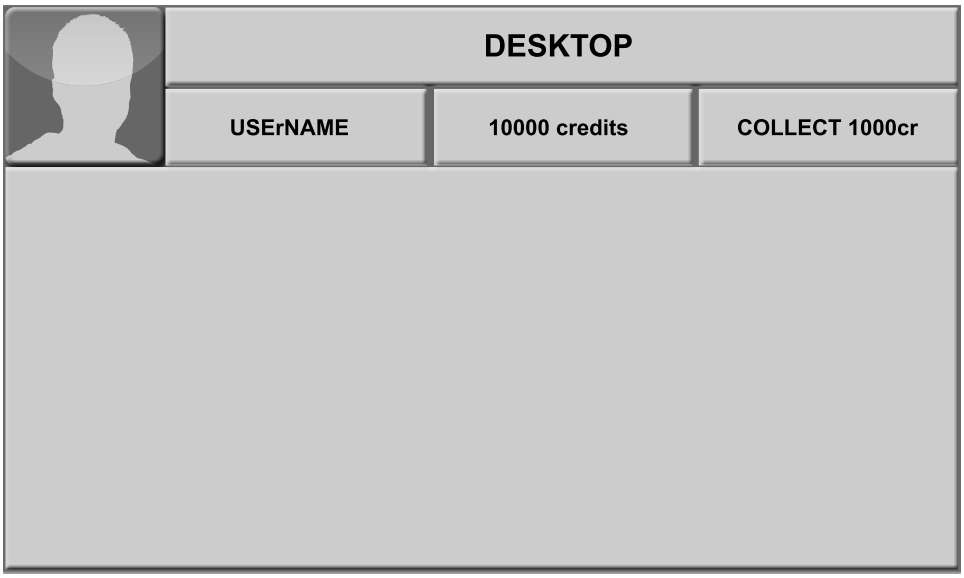

Figure 3.14: Application running on a desktop with a resolution 1920 × 1200px.

wrong or right. Due to the problem statement, the specification and delimitation, and the cooperation with APEX gaming technology GmbH, that set the scope for the implementation, the LibGDX framework was selected, as the cross platform framework to implement the prototype application, from the list frameworks presented in section 2.4.4.

With the mathematical notations, the basis for the application development is formed. Further architecture descriptions give an insight in the 2D scene graph basics of LibGDX, the application design, and the system architecture of the actual implementation, that forms the main part of this chapter. The three most relevant principles adopted from RWD are implemented within this section; fluid grids, flexible images, and Media Queries. Due to the fluid grid approach a flexible layout, that responds according to the given screen form-factor, can be achieved. Images can be scaled dynamically or cropped, so the flexibility of images is guaranteed. With the possibility to query the most relevant device features, shown in 2.4.4, that LibGDX provides, the implementation of Media Queries is also possible. It may be decided, how the application behaves when the screen gets too small or too big. A developer can also respond to the platform the application is running on.

Chapter 4

Conclusions

As established in section 2.1 sales of mobile devices, whether they are smartphones or tablets, increased dramatically between 2008 and 2014 with a forecast to 2015 that predicts that the trend is unbroken. The number of different platforms, and the large number of different screen form-factors available, makes it difficult for developers to cover all devices. Developers have to deal with that diversity of platforms, screen form-factors and technologies the devices support. Over the years, a change in thinking has taken place in web development. The principle of the desktop first approach is overtaken by the mobile first approach stated in section 2.2. Responsive Web Design, explained in section 2.3, has become popular within recent years. Using the principles of Responsive Web Design a larger amount of devices with different screen form-factors can be reached. In fact, the same problem of different screen form-factors appear when developing native or cross platform applications. Adopting and adapting the principles of Responsive Web Design in cross platform development – as explained in section 2.4 – native applications, that fit a large number of screen form-factors, for more than one platform, can be developed. That is exactly what is proposed in the hypothesis in section 1.3.

The allegation the hypothesis establishes says, that it is possible to adopt techniques from Responsive Web Design, for use within the environment of a cross platform framework, to develop applications that run natively on specific devices.

The purpose of the sample application – further readings in section 3.1 – is to verify this hypothesis. With the implementation of the actual prototype application in section 3.4, the hypothesis can be confirmed, because it is possible to adopt the three main techniques of Responsive Web Design – Fluid Grids, Flexible Images, and Media Queries – within the LibGDX framework using Java as programming language.

4.1 Result

It has been shown that the principles of RWD can be used in other frameworks than LibGDX, with the use of another programming language than Java. A precondition is, that at least the three main principles of Responsive Web Design – Fluid Grids, Flexible Images, and Media Queries – can be implemented.

4.2 Implications and Future Research

The use of the terminology *Responsive Web Design*, as a principle for the creation of a website or web application that responds on the screen it is displayed on, has already become popular. In application development no such terminology exists. Assuming the principles of RWD can be adopted, or at least adapted, in every programming language, a new terminology is proposed. *Responsive Application Design* is therefore a more suitable term. Replacing the *web*, of Responsive Web Design, with *application* makes the expression Responsive Application Design more generic. The terminology fits when developing web applications as well as native applications. Thinking of Adaptive Web Design, another principle of modern web design stated in section 2.3.1, raises the question why Responsive Application Design should not be called *Adaptive Application Design*. As already mentioned in section 2.3.1, Adaptive Web Design gets the features of the device it is displayed on, and then the content is sent according to these features. In the name of Adaptive Application Design this means that an official application store, where the application can be downloaded, receives the features of the device it is installed on. This is desirable, but not yet state of the art. Every device that downloads the application over an official application store receives the same application. Although it can be distinguished between a tablet and a smartphone application, but the dimensions of both mobile devices continues to move along. Therefore the terminology of Responsive Application Design is fitting better. The same application gets downloaded on each device, with low and high resolution content, and the application decides, which one is chosen for representation. The designation described in this section is merely a suggestion and therefore needs further research, that would, however, exceed the scope of this thesis.

Appendix A

Terms

ACL	AccessControl List
API	Application Programming Interface
App	Application
CMS	Content Management System
CSS	Cascading Style Sheet
DOM	Document Object Model
dp	Device/Density Independent Pixel
GPS	Global Positioning System
GSM	Global System for Mobile Communications orig. Croupe Special Mobile
HTML	HyperText Markup Language
IDE	Integrated Development Environment
JVM	Java Virtual Machine
OS	Operating System
PDA	Personal Digital Assistant
ppi	Pixels per inch
px	Pixel
QWERTY	Most common modern-day keyboard layout for latin script
RIM	Research In Motion
RWD	Responsive Web Design
SDK	Software Development Kit
Simon	Simon Personal Communicator
UI	User Interface
VM	Virtual Machine
WHATWG	Web Hypertext Application Technology Working Group
W3C	World Wide Web Consortium

Appendix B

Content of the CD-ROM

Format: CD-ROM, Single Layer

B.1 PDF-Files

Path: /

_MasterThesis.pdf	Thesis with instructions (document)

B.2 LaTeX-Files

Path: /

_MasterThesis.tex	Masterthesis (main document)
abstract.tex	Abstract
introduction.tex	Chapter 1
theoretical.tex	Chapter 2
practical.tex	Chapter 3
conclusion.tex	Chapter 4
anhang_a.tex	Anhang A (Terms)
anhang_b.tex	Anhang B (Content of the CD-ROM)
messbox.tex	Messbox zur Druckkontrolle
literatur.bib	Literature database (BibTeX-File)

B.3 Style/Class-Files

Path: /

hgbthesis.cls	LaTeX Class-File for the Masterthesis
hgb.sty	LaTeX Style-File for all Hagenberg documents

hgbbib.sty	LaTeX Style-File for bibliography
hgbheadings.sty	LaTeX Style-File for headings

B.4 Miscellaneous

Path: /res/images

000Android.pdf	Vector graphic (Android design).
000Desktop.pdf	Vector graphic (desktop design).
001.pdf	Vector graphic application (first part).
002.pdf	Vector graphic application (second part).
003.pdf	Vector graphic application (third part).
004.pdf	Vector graphic application (fourth part).
005.pdf	Vector graphic application (fifth part).
006.pdf	Vector graphic application (sixth part).
007.pdf	Vector graphic application (seventh part).
008.pdf	Vector graphic application (eight part).
androidMetrics.pdf	Vector graphic (metrics Android design guidelines).
applicationDesign.pdf	Vector graphic (design of the sample application).
classDiagram.pdf	Vector graphic (class diagram).
sales20082015.pdf	Vector graphic (global device sales 2008 – 2015).
screenFormFactors.pdf	Vector graphic (possible screen form-factors).
libgdxActor.pdf	Vector graphic (class diagram – LibGDX Actor).
webDesign.pdf	Vector graphic (web design).
webdesign.png	Pixel graphic (web designs on smartphones).

Path: /res/literature

*.pdf	Literature, named according to the bibtexkeys of the literature file.
web.zip	Resources of all the downloaded websites.

Path: /res/raw

androidMetrics.psd	Original Adobe Photoshop file (Android metrics).
avatar.psd	Original Adobe Photoshop file (avatar and icon graphics).

prototypeApplication.psd	Original Adobe Photoshop file (all parts of the application).
screenFormFactors.psd .	Original Adobe Photoshop file (possible screen form-factors).
appDesign.graphml . . .	Original yED file (design of the sample application).
classDiagram.graphml .	Original yED (class diagram).
libgdxActor.graphml . .	Original yED (class diagram – LibGDX Actor).
webDesign.graphml . .	Original yED file (web design).

Path: /printVersion

_MasterThesis.pdf . . .	Thesis with instructions, optimized for printing (document).

Path: /sampleApplication

rad_SampleApplication.zip	Source code demo.

References

Literature

[1] Aaron Gustafson. *Adaptive Web Design – Crafting Rich Experiences with Progressive Enhancement*. Easy Readers, LLC, 2011 (cit. on p. 16).

[2] Ethan Marcotte. *Responsive Web Design*. A Book Apart, 2011 (cit. on pp. 15, 16, 18–20).

[3] Jennifer Niederst Robbins. *Learning Web Design – A Beginner's Guide to HTML, CSS, JavaScript and Web Graphicx*. Ed. by 4. [Online; Stand 19. September 2014]. O'Reilly Media Inc., 2012 (cit. on p. 14).

[4] Lawrence Rosen. *Open Source Licensing*. Prentice Hall PTR, 2004, p. 85 (cit. on p. 21).

[5] Luke Wroblewski. *Mobile First*. Ingram, 2011 (cit. on p. 12).

Online sources

[6] *15 top web design and development trends for 2012*. netmag. Jan. 2012. URL: http://www.creativebloq.com/industry-trends/15-top-web-design-and-development-trends-2012-1123018 (cit. on p. 16).

[7] *Adobe Announces Agreement to Acquire Nitobi, Creator of PhoneGap*. Adobe. Oct. 2011. URL: http://www.adobe.com/aboutadobe/pressroom/pressreleases/201110/AdobeAcquiresNitobi.html (cit. on p. 24).

[8] Android. *Android Design*. Android. URL: https://developer.android.com/design/index.html (cit. on p. 32).

[9] Android. *Android, the world's most popular mobile platform*. Android. URL: https://developer.android.com/about/index.html (cit. on p. 7).

[10] Syed Tariq Anwar. *NTT DoCoMo and M-Commerce – A Case Study in Market Expansion and Global Strategy*. NTT DoCoMo. 2001. URL: http://www.itu.dk/~rold/1_sem/B1/Cases/DoCoMo.pdf (cit. on p. 5).

[11] *Apache Cordova gets a new look*. h-online. Feb. 2012. URL: http://www.h-online.com/open/news/item/Apache-Cordova-gets-a-new-look-1440114.html (cit. on p. 24).

[12] APEX. *APEX by name APEX by nature*. APEX gaming technology GmbH. URL: http://apex-gaming.com/about-apex/ (cit. on p. 30).

[13] Apple. *App Review*. Apple Inc. URL: https://developer.apple.com/app-store/review/ (cit. on pp. 6, 7).

[14] Apple. *App Store Downloads Top 100 Million Worldwide*. Apple Inc. Sept. 2008. URL: http://www.apple.com/pr/library/2008/09/09App-Store-Downloads-Top-100-Million-Worldwide.html (cit. on p. 6).

[15] Apple. *App Store Tops 40 Billion Downloads with Almoast Half in 2012*. Apple Inc. Jan. 2013. URL: http://www.apple.com/pr/library/2013/01/07App-Store-Tops-40-Billion-Downloads-with-Almost-Half-in-2012.html (cit. on p. 7).

[16] Apple. *Apple Reinvents the Phone with iPhone*. Apple Inc. Sept. 2007. URL: http://www.apple.com/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html (cit. on p. 6).

[17] *Apple Info*. Apple Inc. URL: https://www.apple.com/about (cit. on pp. 6, 7).

[18] *Betriebssystem: Sony UI*. German. Androidhandys. URL: http://androidhandys.de/sony-ui.html (cit. on p. 8).

[19] BlackBerry. *BlackBerry Charts new Course by Officially Adopting its Iconic Brand Name*. BlackBerry. July 2013. URL: http://press.blackberry.com/press/2013/blackberry-brand-name.html (cit. on p. 5).

[20] *Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web*. Twitter. URL: http://getbootstrap.com/ (cit. on p. 21).

[21] Ryan Boudreaux. *What is the difference between responsive vs. adaptive web design?* TechRepublic. Apr. 2013. URL: http://www.techrepublic.com/blog/web-designer/what-is-the-difference-between-responsive-vs-adaptive-web-design/ (cit. on p. 16).

[22] Pete Cashmore. *Why 2013 is the Year of Responsive Web Design*. Mashable. Dec. 2012. URL: http://mashable.com/2012/12/11/responsive-web-design/ (cit. on p. 16).

[23] Thomas Claburn. *Google's Secret Patent Portfolio Predicts gPhone*. InformationWeek. Sept. 2007. URL: http://www.informationweek.com/googles-secret-patent-portfolio-predicts-gphone/d/d-id/1059389?cid=nl_iwk_daily (cit. on p. 8).

[24] *Corona SDK — The ultimate 2D development platform*. Corona Labs Inc. URL: http://coronalabs.com/ (cit. on pp. 24–26).

[25] *CSS APIs Current Status*. W3C. URL: http://www.w3.org/standards/techs/js#w3c_all (cit. on p. 15).

[26] *CSS current work and how to participate*. W3C. URL: http://www.w3.org/Style/CSS/current-work (cit. on p. 15).

[27] *CSS3 Introduction*. W3School. URL: http://www.w3schools.com/css/css3_intro.asp (cit. on p. 15).

[28] *Definition of responsive in English*. Oxford Dictionaries. URL: http://www.oxforddictionaries.com/definition/english/responsive (cit. on p. 13).

[29] Matt Doyle. *Responsive Web Design Demystified*. Elated. Sept. 2011. URL: http://www.elated.com/articles/responsive-web-design-demystified/ (cit. on p. 16).

[30] *Easily create apps using the web technologies you know and love: HTML, CSS, and JavaScript*. Adobe. URL: http://phonegap.com (cit. on pp. 24–26).

[31] Ben Elgin. *Google Buys Android for Its Mobile Arsenal*. Businessweek. Aug. 2005. URL: http://www.businessweek.com/stories/2005-08-16/google-buys-android-for-its-mobile-arsenal (cit. on p. 8).

[32] Ericsson. *Ericsson GS88 Preview*. URL: http://pws.prserv.net/Eri_no_moto/GS88_Preview.htm (cit. on p. 5).

[33] *Foundation — The most advanced responsive front-end framework in the world*. Zurb. URL: http://foundation.zurb.com/ (cit. on p. 21).

[34] Tobias Gebauer. *Die 10 besten responsive Frameworks*. German. The-Webdesign. Apr. 2013. URL: http://www.the-webdesign.net/die-besten-10-responsive-frameworks/ (cit. on pp. 20, 21).

[35] Ronen Halevy. *The History of RIM and the BlackBerry Smartphone, Part 3: The Evolution Of Color*. berry review. Mar. 2009. URL: http://www.berryreview.com/2009/03/16/the-history-of-rim-the-blackberry-smartphone-part-3-the-evolution-of-color/ (cit. on p. 5).

[36] Eva Harb et al. *Responsive Web Design*. TU Graz. 2011. URL: http://courses.iicm.tugraz.at/iaweb/surveys/ws2011/g3-survey-resp-web-design.pdf (cit. on pp. 10–12).

[37] Greg Hickman. *What Small Businesses Need To Know About Mobile Marketing*. URL: http://mobilemixed.com/what-small-businesses-need-to-know-about-mobile-marketing/ (cit. on p. 10).

[38] *HTC Sense*. German. Androidhandys. URL: http://androidhandys.de/htc-sense.html (cit. on p. 8).

[39] *HTML — Living Standard*. WHATWG. Sept. 2014. URL: http://www.whatwg.org/specs/web-apps/current-work/multipage/ (cit. on p. 14).

[40] *HTML5 — A vocabulary and associated APIs for HTML and XHTML – W3C Candidate Recommendation*. W3C. Dec. 2012. URL: http://www.w3.org/TR/2012/CR-html5-20121217/ (cit. on p. 14).

[41] *HTML5 Introduction*. W3School. URL: http://www.w3schools.com/html/html5_intro.asp (cit. on p. 14).

[42] *Industry Leaders Announce Open Platform for Mobile Devices*. Open Handset Alliance. Nov. 2007. URL: http://www.openhandsetalliance.com/press_110507.html (cit. on p. 8).

[43] Daniel IoPCmag. *Original Android Prototype Revealed During Google, Oracle Trial*. PCWorld. Apr. 2012. URL: http://www.pcworld.com/article/254539/original_android_prototype_revealed_during_google_oracle_trial.html (cit. on p. 8).

[44] *Java Language and Virtual Machine Specifications*. Oracle. URL: http://docs.oracle.com/javase/specs/ (cit. on p. 22).

[45] *JavaScript Tutorial*. W3School. URL: http://www.w3schools.com/js/default.asp (cit. on p. 15).

[46] Dirk Jesse. *Flexible Layouts: Challenge For The Future*. Smashing-Magazine. June 2008. URL: http://www.smashingmagazine.com/2008/06/26/flexible-layouts-challenge-for-the-future (cit. on p. 18).

[47] Athanassios Kaliudis. *TouchWiz, Sense und co.: Android-UIs im Vergleich*. German. Connect. Jan. 2014. URL: http://www.connect.de/ratgeber/touchwiz-sense-android-uis-im-vergleich-1489525.html (cit. on p. 8).

[48] *LibGDX*. Badlogicgames. URL: http://libgdx.badlogicgames.com/ (cit. on pp. 25, 26, 33).

[49] Linfo. *Cross-Platform Definition*. The Linux Information Project. Dec. 2005. URL: http://www.linfo.org/cross-platform.html (cit. on pp. 22, 23).

[50] Kevin Lynch. *The Multiscreen Revolution*. Adobe Inc. Feb. 2011. URL: http://blogs.adobe.com/conversations/2011/02/the-multiscreen-revolution.html (cit. on p. 12).

[51] MarketWatch. *Annual Financials for Apple Inc.* The Wall Street Journal. Sept. 2014. URL: http://www.marketwatch.com/investing/stock/aapl/financials (cit. on p. 7).

[52] *Mobiltelefone; Handys ohne Vertrag*. URL: https://geizhals.at/?cat=umtsover&xf=149_Touchscreen#xf_top (cit. on p. 10).

[53] Chris Moor. *T-Mobile G1 Event Round-up*. Talkingandroid.com. Sept. 2008. URL: http://www.talkandroid.com/260-t-mobile-g1-details/ (cit. on pp. 6, 8).

[54] Nokia. *Nokia in 2007*. Nokia. 2007. URL: http://company.nokia.com/sites/default/files/download/05-nokia-in-2007-pdf.pdf (cit. on p. 6).

[55] PCmag. *Definition of: cross platform*. PCmag. URL: http://www.pcmag.com/encyclopedia/term/40495/cross-platform#fbid=aHfb3ldkqPq (cit. on p. 23).

[56] PCmag. *Definition of: tablet computer*. PCmag. URL: http://www.pcmag.com/encyclopedia/term/52520/tablet-computer (cit. on p. 5).

[57] PhoneScoop. *Feature Phone (Definition)*. phone scoop. URL: http://www.phonescoop.com/glossary/term.php?gid=310 (cit. on p. 4).

[58] PhoneScoop. *Smartphone (Definition)*. phone scoop. URL: http://www.phonescoop.com/glossary/term.php?gid=131 (cit. on pp. 4, 5).

[59] Michael Rougeau. *Google details Android 4.4 KitKat, its latest mobile upgrade*. Rechradar. Oct. 2013. URL: http://www.techradar.com/news/software/operating-systems/google-details-android-4-4-kitkat-its-latest-mobile-upgrade-1195177 (cit. on p. 9).

[60] Ira Sager. *Before iPhone and Android Came Simon, the First Smartphone*. Businessweek. June 2012. URL: http://www.businessweek.com/articles/2012-06-29/before-iphone-and-android-came-simon-the-first-smartphone (cit. on p. 5).

[61] Samer and Chipalo. *WPF Text Measurement Units*. MSDN. 2009. URL: http://blogs.msdn.com/b/text/archive/2009/12/11/wpf-text-measurement-units.aspx (cit. on p. 32).

[62] Eric Schmidt. *Eric Schmidt at DLD*. englis. Jan. 2011. URL: http://www.youtube.com/watch?v=-IGfBGHDHRc#t=587 (cit. on p. 12).

[63] *Startup Design Framework — Suit Up Your Startup*. Designmodo. URL: http://designmodo.com/startup/ (cit. on p. 21).

[64] *TouchWiz von Samsung*. German. Androidhandys. URL: http://androidhandys.de/touchwiz.html (cit. on p. 8).

[65] Christina Warren. *The Pros and Cons of Cross-Platform App Design*. Mashable. Feb. 2012. URL: http://mashable.com/2012/02/16/cross-platform-app-design-pros-cons/ (cit. on pp. 23, 24).

[66] WebDesignShock. *Responsive Web Design, Most Complete Guide*. Sept. 2013. URL: http://www.webdesignshock.com/responsive-web-design/ (cit. on pp. 13, 17).

[67] Peter Weichsel et al. *At Home in the Cloud – The Emerging Opportunity for Telecom Operators*. Booz and Company Inc. 2012. URL: http://www.strategyand.pwc.com/media/file/Strategyand_At-home-in-the-cloud.pdf (cit. on p. 9).