Metropolia

Joni Korpi

Adaptive Web Design

As applied to the design process of a web application

Metropolia University of Applied Sciences
Bachelor of Engineering
Media Technology
Bachelor's Thesis
16.3.2012

Metropolia

Abstract

Author	Joni Korpi
Title	Adaptive Web Design As applied to the design process of a web application
Number of Pages	36 pages
Date	18.3.2012
Degree	Bachelor of Engineering
Degree Programme	Media Technology
Specialisation option	Digital Media
Instructors	Aarne Klemetti, Senior Lecturer Harri Airaksinen, Principal Lecturer

This thesis explored the usage of adaptive web design techniques in the design process of a reservation and inventory management web application, New Reserve. It attempted to uncover issues a designer is likely to face when moving from a traditional web design process to an adaptive one.

Most issues uncovered were related to keeping visual design appealing and attempting to support multiple input methods, such as touch screens and mice, at the same time.

Using a fluid grid for visual design was found to be difficult, as they caused problems when elements start stretching beyond their optimal maximum dimensions, and when element dimensions are determined in a mix of percentage units and absolute units. Intentionally leaving empty space in wide designs and using the alternative "border-box" model were found to alleviate these problems.

Using the "Mobile First" approach for design was found to be recommended, as the amount of mobile internet users is set to overtake desktop internet users very soon. The approach also helps in keeping designs cruft-free. Making images adapt to mobile sizes and bandwidth restrictions was found to be difficult, but there is hope for a standards-based technique to deal with this in the near future.

Using progressive enhancement was found to be the key in developing a web application with features accessible by a wide array of devices. Forcefully adding support for features using polyfills was found to require caution, as a single polyfill may not be suitable for all input methods.

Keywords	adaptive, responsive, web design, grid, mobile, progressive enhancement, polyfill

Metropolia

Tiivistelmä

Tekijä Otsikko Sivumäärä Aika	Joni Korpi Mukautuva web-suunnittelu sovellettuna web-palvelun suunnitteluprosessiin 36 sivua 18.3.2012
Tutkinto	insinööri (AMK)
Koulutusohjelma	mediatekniikka
Suuntautumisvaihtoehto	digitaalinen media
Ohjaajat	lehtori Aarne Klemetti koulutuspäällikkö Harri Airaksinen

Insinöörityössä tutkittiin mukautuvan web-suunnittelun käyttöä web-pohjaisen varaus- ja inventaariopalvelun suunnitteluprosessissa. Pyrittiin erityisesti tuomaan esille niitä prosessissa ilmenneitä ongelmia, joihin web-suunnittelijat saattavat törmätä siirtyessään perinteisemmästä web-suunnitteluprosessista mukautuvaan suunnitteluun. Käytiin myös läpi, miten näihin ongelmiin suhtauduttiin palvelun suunnittelussa.

Suurin osa ilmenneistä ongelmista liittyi joko visuaalisen suunnittelun pitämiseen ilmeikkäänä tai erilaisten osoitinlaitteiden ja kosketusnäyttöjen yhtäaikaiseen tukemiseen.

Joustavien palstojen käyttäminen visuaalisessa suunnittelussa todettiin vaikeaksi, koska ne aiheuttavat ongelmia, kun sivun sisältö alkaa joustaa yli sille suotuisten leveyksien tai korkeuksien. Tämän vuoksi todettiin olevan suotuisaa jättää leveämmillä ruuduilla reilusti tilaa tyhjäksi. Vaikeuksia ilmeni myös silloin, kun joustavien palstojen kanssa käytettiin sekä prosenttiyksiköitä että absoluuttisia yksiköitä. Tähän auttoi vaihtoehtoisen border-box-mallin käyttö.

"Mobiili ensin" -suunnittelutavan käyttöä suositeltiin, koska se auttaa pitämään sivut yksinkertaisina ja sisältönsä ytimeen keskittyneinä. Mobiilikäyttäjien jatkuvasti kasvava määrä on myös hyvä syy keskittyä mobiilisuunnitteluun. Kuvien mobiililaitteisiin mukauttamisen todettiin olevan nykymenetelmillä vaikeaa, mutta lähitulevaisuudessa ongelman luultavasti ratkaisee uusi standardeihin perustuva menetelmä.

Progressiivisen ehostuksen käytön todettiin olevan tärkeää mukautuvien web-palveluiden kehittämisessä, mutta siihen liittyvien web-selainten toiminnallisuutta paikkaavien kirjastojen käyttämisen todettiin vaativan varovaisuutta, koska sama kirjasto ei välttämättä ole sopiva kaikille mahdollisille osoitinlaitteille ja käyttöjärjestelmille.

Avainsanat	mukautuva, adaptiivinen, responsiivinen, web-suunnittelu, palsta, mobiili, progressiivinen ehostus, polyfill

Table of contents

Terms and abbreviations

adaptive web design
: A set of techniques for making websites that adapt in different ways to the capabilities and features of the browser being used to access them.

responsive web design
: An adaptive web design technique, consisting of the use of fluid grids, fluid media, and media queries.

media query
: A conditional clause used to restyle web pages depending on the capabilities of the browser used to access it.

fluid grid
: A design structure composed out of percentage units.

fixed grid
: A design structure composed out of absolute units, like pixels or ems.

CSS
: *Cascading Stylesheets.* The language used to style web pages.

em
: A CSS unit relative to the "font-size" of its element.

HTML5
: *Hypertext Markup Language 5.* The most recent version of the markup language used to write web pages.

W3C
: *World Wide Web Consortium.* A web standard development community.

UI
: *User interface.*

1. Introduction

Adaptive web design is a new paradigm for designing websites. It deals with the fact that websites are no longer viewed only on monitors of a certain size, but on screens of innumerable different sizes, ranging from tiny mobile phones to 50" widescreen televisions. Unlike traditional fixed-width websites, good adaptive websites reshape themselves to fit screens of almost any size, while making sure the site's content remains legible and its context appropriate.

At the time of writing, much experimentation has been done with adaptive design in the field of traditional content-driven websites, most notably in the form of "responsive web design", a design method coined by Ethan Marcotte. When it comes to websites that are more like applications than just containers for content, not much experimentation or research has surfaced yet. [1.]

This thesis aims to discuss some of the problems that became apparent when using adaptive and responsive design methods in the redesign of an inventory and reservation management web application for Metropolia's Media Lab. The topics will focus around those phases of the design process where adaptive design methods caused problems or required changes to the traditional process or its methods.

2. The New Reserve project

The existing reservation system (Old Reserve) was a fairly straightforward but visually complex web application. Its visual and interaction design (or lack thereof) made it difficult to operate for both of its user groups: the regular users and the administrators.

Regular users were primarily able to perform two tasks: browse equipment listed in the system, and submit requests to reserve those items. The user interface made both of these tasks a chore at best, and users were often seen expressing great frustration over the system.

Administrators were supposed to be able to manage the equipment listed in the system and deal with the reservation requests submitted by regular users. The latter worked out so poorly that the whole process was eventually pushed aside and users

were asked to print out their reservation requests on paper, sign them, and deliver them to the administrator by hand.

In the redesign of this system (New Reserve), my teammate Jukka Lähetkangas (the back-end developer) and I (the designer and front-end developer) set out to fix the aforementioned failings of Old Reserve. We opted for a complete redesign and rewrite, instead of attempting to evolve the existing system.

For the administrators, we attempted to accomplish the following:

- Moving administration tools "inline"; instead of burying common administration controls on separate pages, they'd be mostly integrated into the same views as regular users see. For example: when viewing an item, an administrator would see "Edit" and "Delete" buttons right next to the item, while a regular user would not. This should make the application more intuitive to use for both new and experienced administrators.
- Letting administrators add images, descriptions and guides to each item, making it easier for regular users to identify and eventually use the items.
- Giving administrators the option to hide items from regular users, by tagging them "Unreservable". This would let New Reserve double as a proper inventory management system.
- Adding a blog of sorts on the front page of the application, which would let administrators make important announcements and alert users when needed.
- Enabling administrators to finally manage, accept and decline reservations within the application, without resorting to using pen and paper.

For the benefit of the regular users, our aims included:

- Simplifying equipment browsing with clear categorization.
- Speeding up the reservation process by adding a "shopping cart", letting users reserve several items at the same time, as well as by decreasing the amount of steps in the reservation process.
- Making it obvious which items are reservable and not reservable at any given time.
- Creating a visual design that follows Metropolia's brand guidelines.

- Removing the need for regular users to register a new account or fill in their personal details for every new reservation, by using Metropolia's own authentication system.
- And most importantly: making the web application accessible from a wide array of different kinds of devices, which is what this thesis concentrates on.

3. How adaptive design works

Adaptive web design is about making websites that adapt to various attributes of the web browser they are being accessed with, be it screen size or technical capabilities. The most prominent form of adaptive web design, at the time of writing, is "responsive web design", which is comprised of three components: media queries, a fluid grid, and fluid media. (The latter two will be discussed later in this thesis.) [1.]

Media queries are essentially questions that the website can submit to a web browser. These questions can be used to determine the browser's attributes, such as dimensions, aspect ratio, pixel density and colour support. When these attributes are known, the website can adapt to them by, for example, adjusting its font size or rearranging elements in its layout. [1, 2.]

```
@media screen and (min-width: 1280px) {
      body {
            font-size: 110%;
      }
}

@media screen and (max-width: 320px) {
      #content {
            max-width: 320px;
      }
}
```

Code example 1.

In the example above, the first block of code contains a media query asking if the browser window is at least 1280 pixels wide. If the query returns true, the overall font-size of the page will increase by 10%. The second block of code queries whether the browser is less than or exactly 320 pixels wide and adjusts the width of an element in the layout to be no wider than 320 pixels.

The examples above were CSS code, but media queries can also be made in Javascript, with the help of the Modernizr library [3]. This allows the website to dynamically shape not only its layout, but its entire structure and content as well, allowing, in theory, the website to reshape itself in endless ways depending on what kind of a browser and device it is being viewed with.

```
if ( Modernizr.mq('screen and (min-width: 1280px)') ) {
      // Reshape things to be
      // more useful on large screens
}
```

Code example 2.

Depending on who you ask, adaptive design can cover a variety of other design methods as well, such as progressive enhancement, responsive server-side methods, and more. However, for the purposes of this thesis, out of all the technical methods used in adaptive design, only understanding media queries — and the later explained fluid media and fluid grids — is important.

4. The adaptive design process

To understand the choices made and the problems encountered during New Reserve's adaptive design process, one must first understand how making adaptive websites differs from making traditional ones. The following chart shows my rough approximation of a traditional web design process.

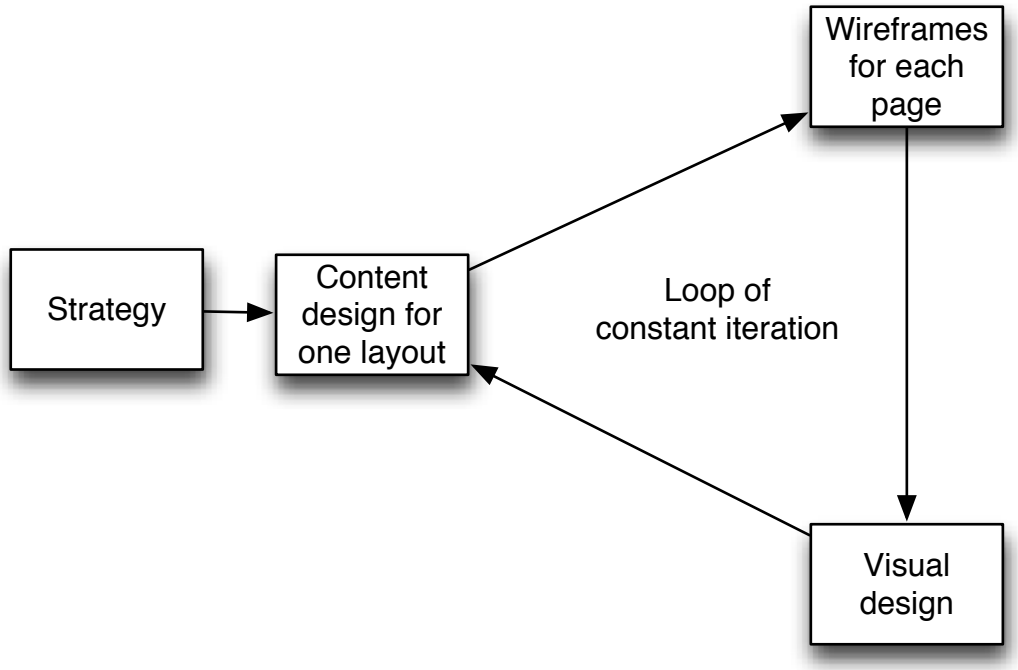

Figure 1. A traditional web design process.

Web design — when dumbed down into a flowchart — is a fairly simple iterative process. The first step, strategy, is followed a loop of constant iteration, containing several other steps. The process has no clear end point, since it often continues throughout the website's lifespan.

In an adaptive web design process, the core concepts do not really change; they just become more complex.

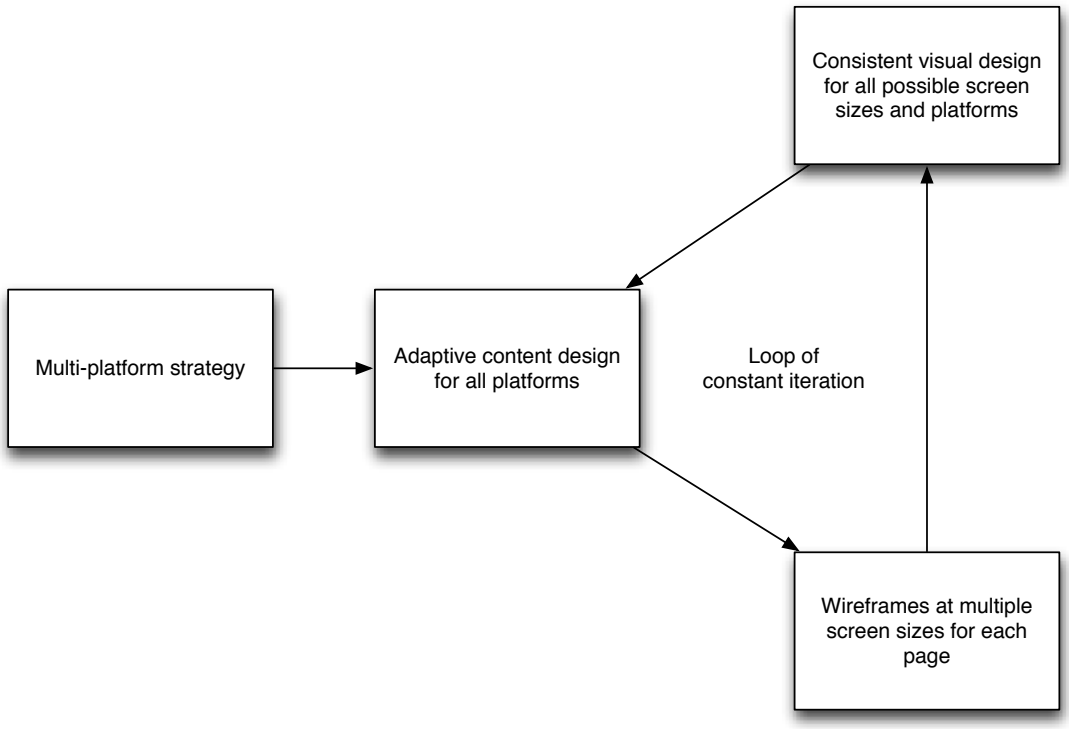

Figure 2. An adaptive web design process.

Strategy still informs the rest of the process, but it has become more complex, requiring planning for all possible types of devices. Each step of the iterative loop has also become more complex, requiring more work and skill to complete. Some other aspects also add to the complexity:

- There are currently no proper tools for designing neither interactive user interfaces nor ones that adapt to different screen sizes and contexts. Work has to be done with pen and paper, and real-life interactive prototypes.
- Each step of the process requires more work, making the process slower and costlier.
- Not only are more skills required from the designer, but also new skills; traditional print or screen design techniques do not apply very well to designing for varying screen sizes.

As difficult as the process sounds, personally I would say the results of are worth the added complexity and difficulty. There are many technical benefits, which will be discussed later in this thesis, but also many non-technical benefits:

- Building just one version of the site, instead of, for example, a separate mobile and tablet site comes with a huge benefit. Having just one codebase to worry about makes life easier for everyone involved with the site, and makes the site easier to access and understand for users.
- Since all shapes and sizes of the site are designed as one, it will appear and feel consistent across different kinds of devices.
- Adaptive websites are "future-friendly". Since they're designed to be viewed on screens of any size and shape, they should also work well on any future screens; adaptive websites embrace the unpredictability of the web, instead of trying to fight it. [4.]

5. Graphic design

5.1. Choosing a method of adapting

Perhaps the most fundamental decision to make in designing an adaptive website is to choose between a fluid-width or a fixed-width grid system. A grid system is a set of vertical and horizontal guides, which can be used to determine the sizes and positions of various elements in a design in a consistent manner. For the purposes of keeping this chapter easy to understand, I will be concentrate on the vertical guides.

Fluid-width grid systems, or "fluid grids", are part of responsive web design. In a fluid grid, the positions of guides are determined relative to the browser window's width. [5.]

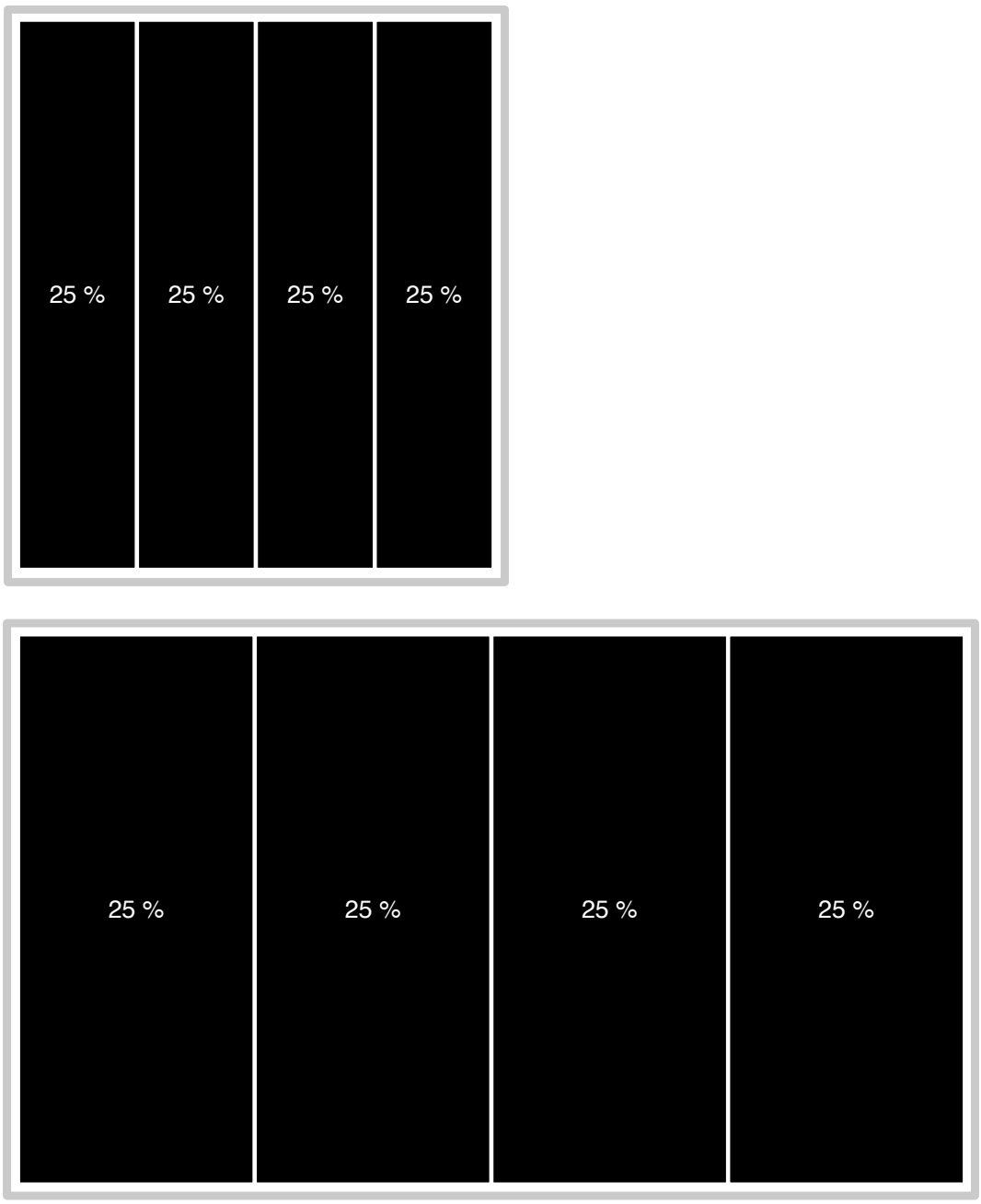

Figure 3. A fluid-width grid, divided into four parts, each 25% of the browser window's width.

Sites designed using a fluid-width grid will adapt to the screen's width pixel by pixel, with the design elements stretching to fill up the available space. This only works at certain ranges of widths though, since most elements cannot be stretched infinitely without negative impact; paragraphs of text cannot be too wide or too narrow, or their readability will suffer [6], while images cannot be stretched beyond their native resolutions without loss in quality, and on widescreen-shaped screens there is the danger of

filling up too much vertical space with a single image. Because of this, fluid grids have to often be "frozen" at certain screen widths to prevent them from stretching any wider, essentially turning them into fixed grids after a certain screen width has been reached. [5, 7.]

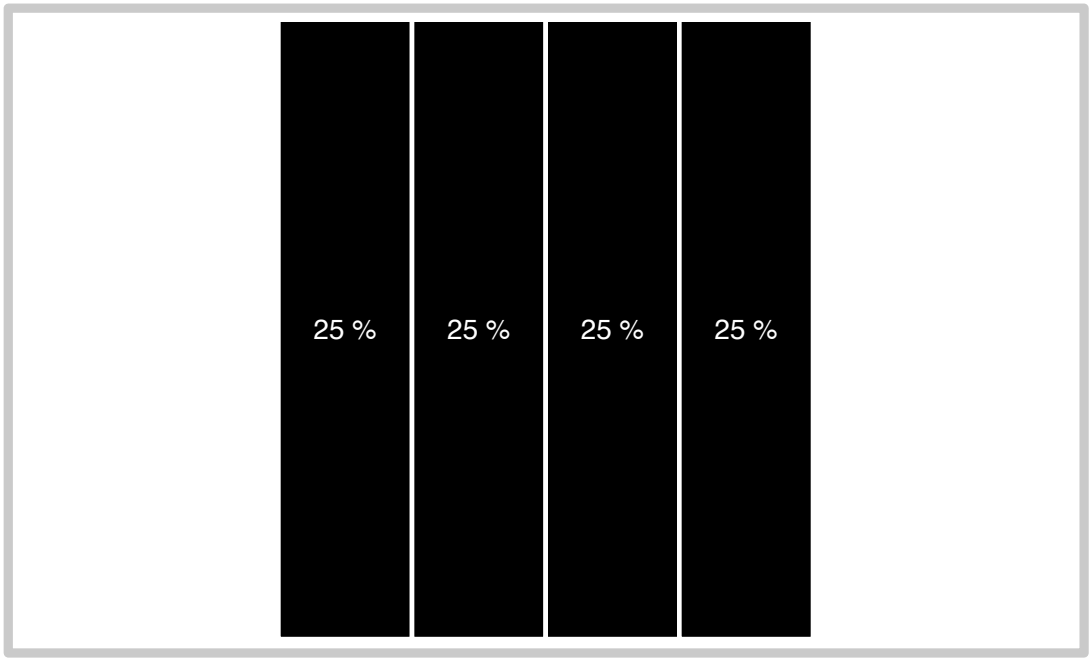

Figure 4. A "frozen" fluid grid with empty space on the sides.

In fixed-width grid systems, or "fixed grids", column widths are set in pixels or ems (one "em" equals the element's font size in pixels). If set in ems, the widths will stay in proportion with the site's base font size, and can therefore be scaled up and down in proportion the site's content simply by changing font sizes. This is especially good for the site's accessibility, since users can manually override font-sizes to compensate for bad eyesight, for example. In this scenario, a fixed grid set in ems will simply scale up and down, always staying in proportion with the content's font sizes. This is also called "elastic" behaviour. [7.]

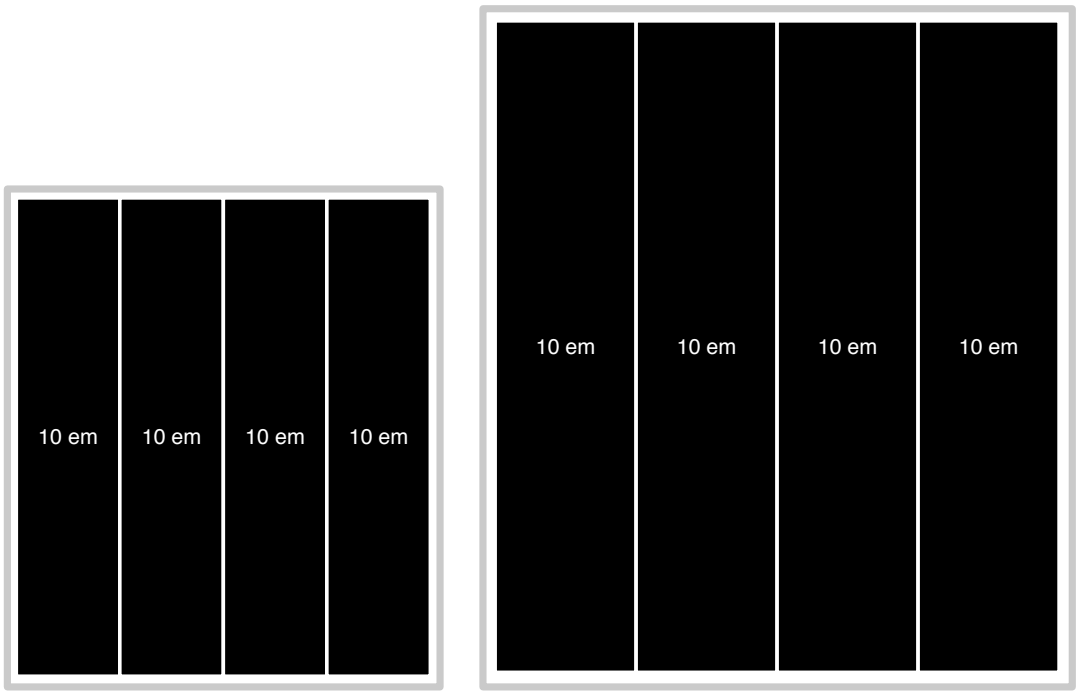

Figure 5. An "elastic" fixed grid, with the base font size increased on the right.

If a fixed grid is set in pixels, its columns will generally stay in proportion to the screen's actual physical pixels, unless the viewport (the "canvas" inside the browser window, in which the site appears) is zoomed in or out, in which case the grid zooms with it. Today, especially many mobile browsers do this kind of zooming automatically. The general consensus among web designers today is that ems are recommended over pixels, since they improve accessibility by staying intact when users change their font sizes, and have very few negative sides compared to pixels. [7.]

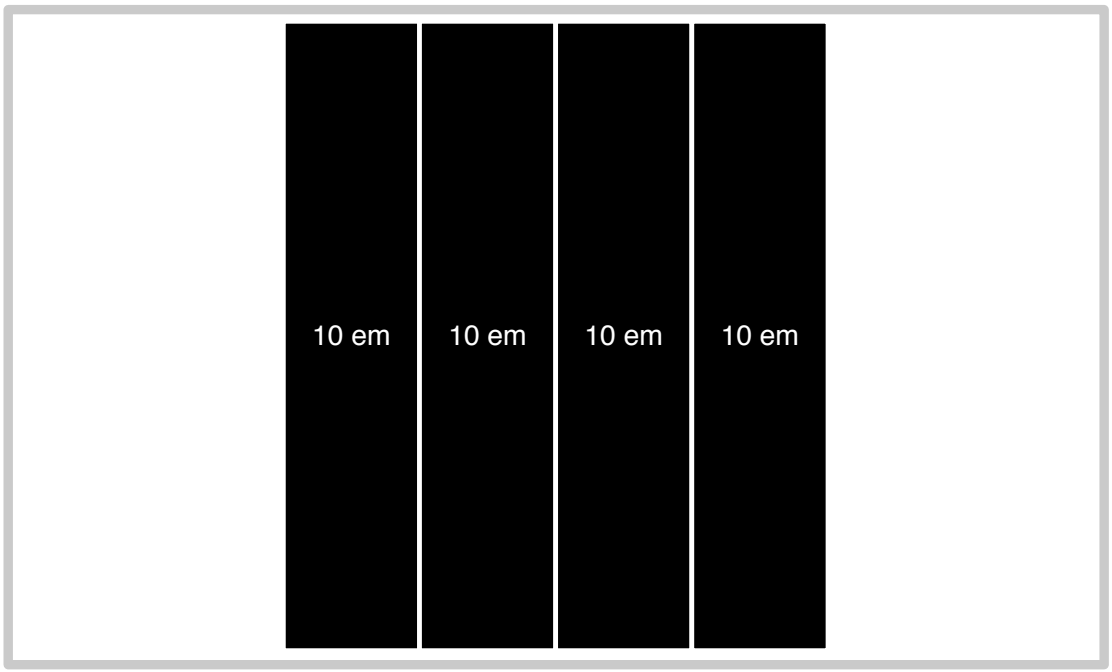

Figure 6. A fixed-width grid centered on the screen, with empty space on the sides. It looks much like the aforementioned "frozen" fluid grid.

When a fixed-width grid is used in adaptive design, guides are added or subtracted as the screen size changes, but the spaces between them (also known as "columns") generally do not change, which helps in keeping the site's design feel consistent at different screen sizes.

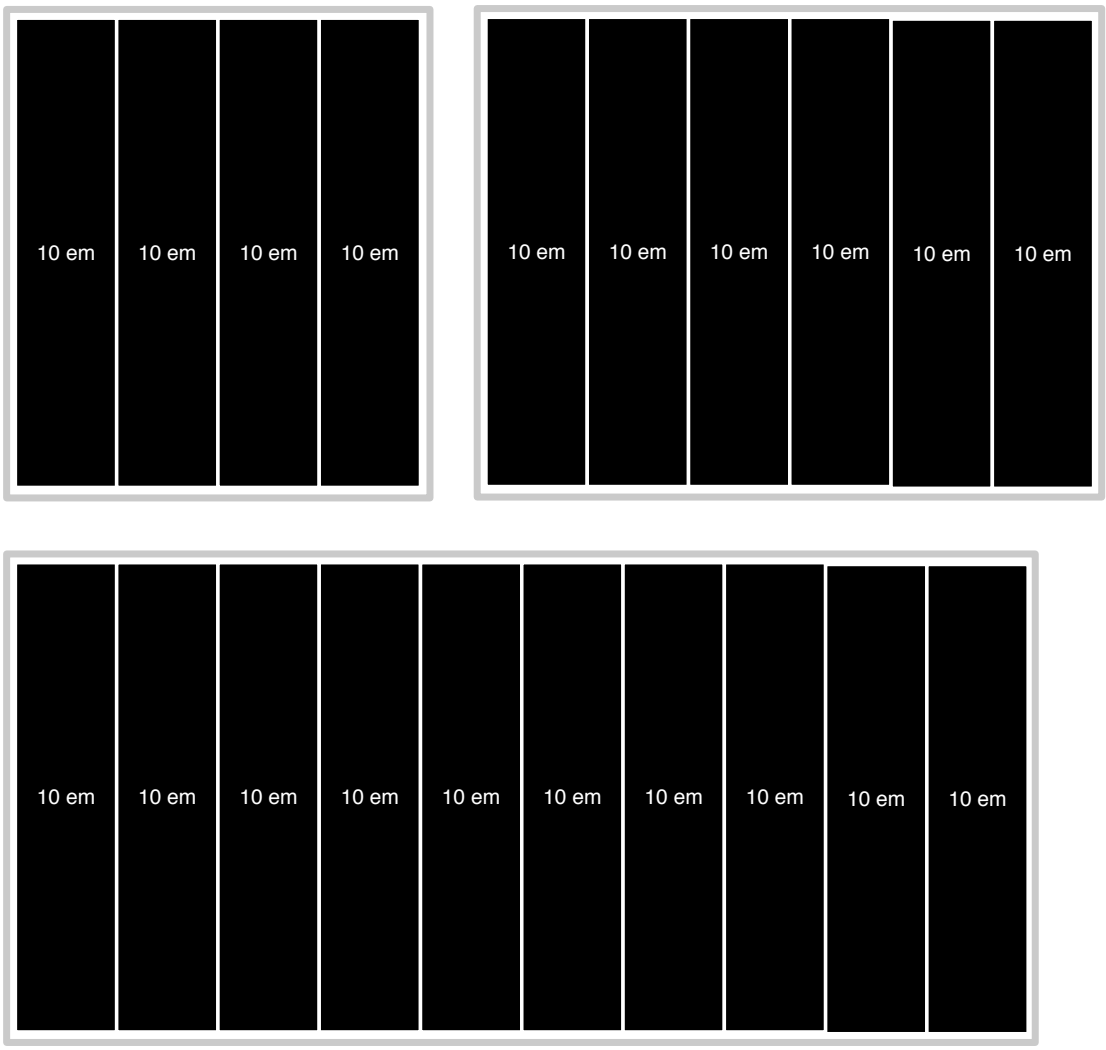

Figure 7. An adapting fixed grid.

Usually, consistency in designs that use fluid grids has to be achieved in other ways. Since the sizes of columns in fluid grids are always in flux, less attention is usually paid to their dimensions when they adapt; a fluid grid divided into four 25 % wide columns may suddenly change into one with five 20 % wide columns, which can feel disorienting. One way I have personally found to mitigate this effect in fluid grids is to split and combine columns, instead of arbitrarily adding or subtracting them. [8.]

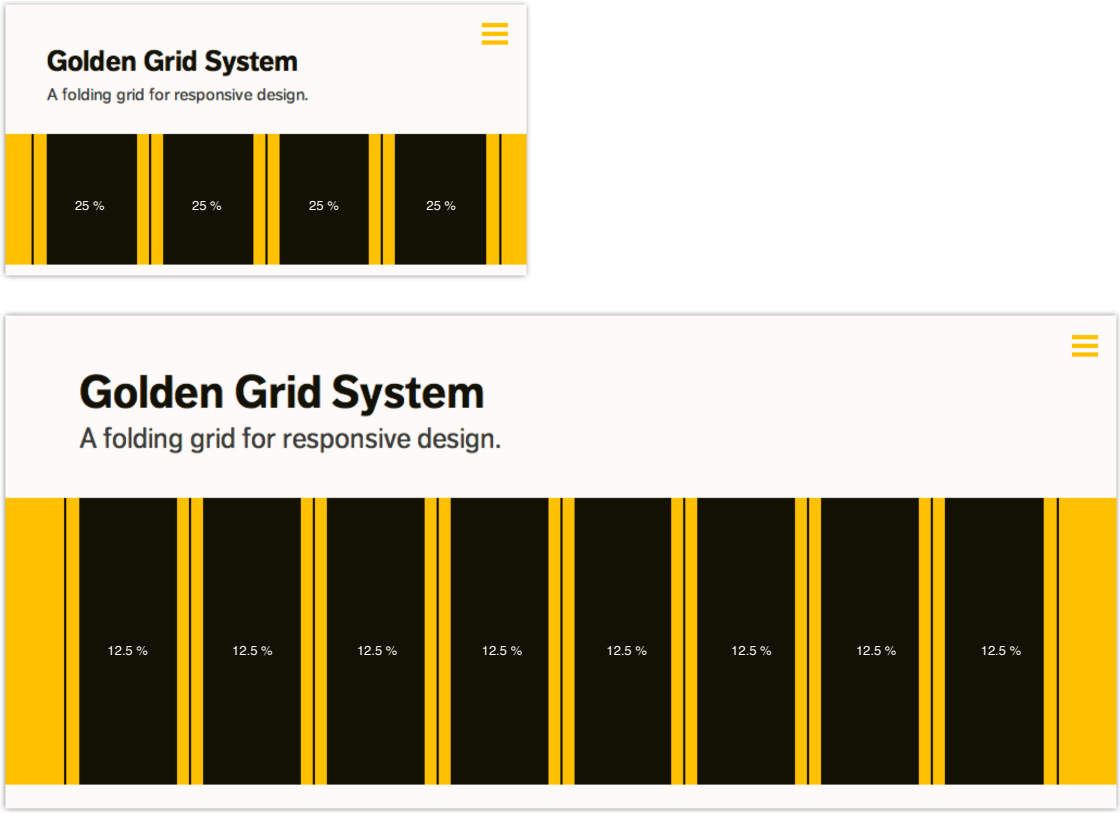

Figure 8. Splitting and combining columns in "Golden Grid System", a design tool published by yours truly [8].

In the design of New Reserve, I ended up using combination of both fluid and fixed-width grids. The narrowest state of the layout is fluid until a certain width is reached. As the layout grows wider, it morphs into a fixed grid to suit tablet-sized screens, which eventually gains more columns to suit desktop-sized screens.

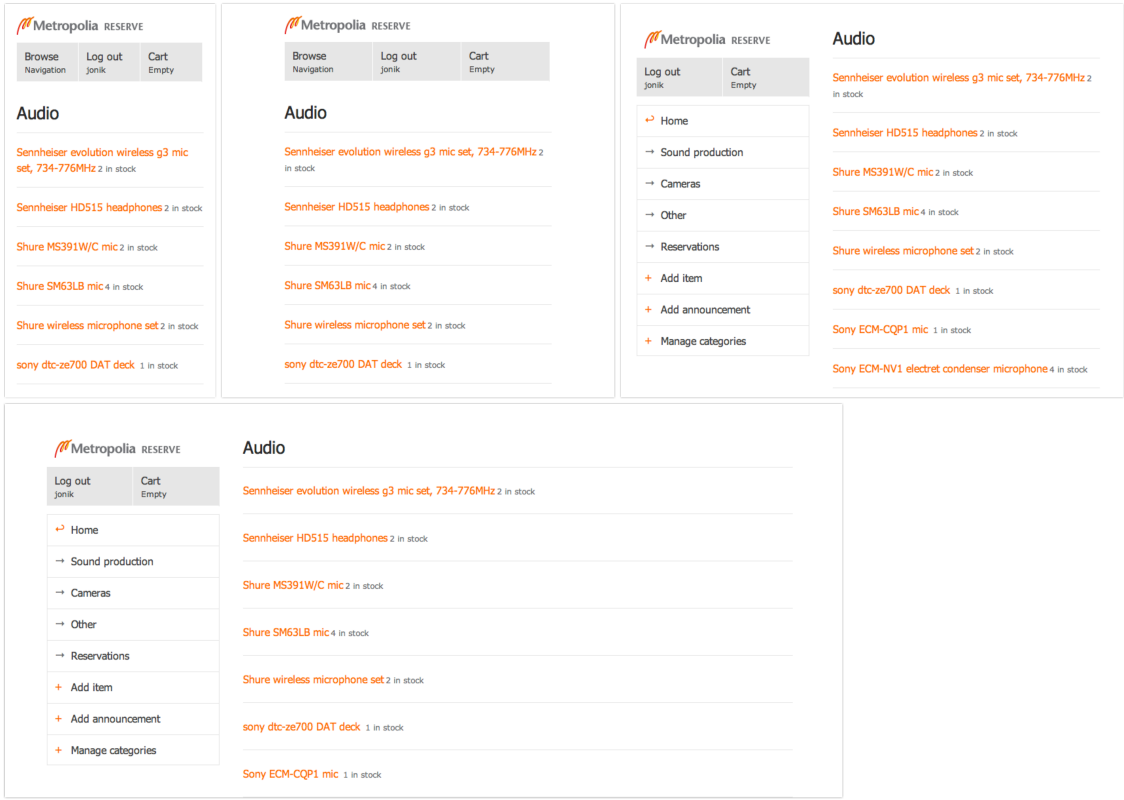

Figure 9. The New Reserve grid in use.

I chose the fixed-width grid for two reasons. First, I was inexperienced in designing web applications, and was worried styling a large amount of navigation and form elements to be fluid on larger screens would be difficult. So, most elements in the design were text-based, making them more difficult to handle in a fluid layout. One exception was the "timeline calendar", which displays current and future reservations for the selected item.

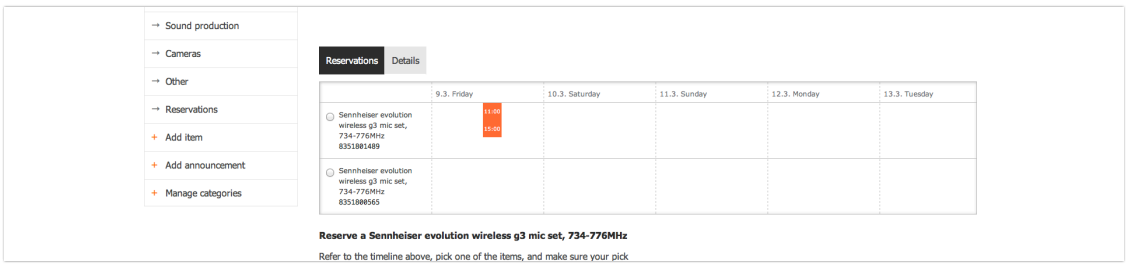

Figure 10. The New Reserve timeline calendar.

The timeline is fully horizontal, and can be infinitely wide, which is why it is inside a horizontally scrollable element. Its theoretically infinite horizontal dimensions would

have made it a perfect candidate to be fit in a fluid grid, since it could even have stretched out to fill the full width of a a 27" desktop monitor. Sadly, I had to compromise and let the timeline element's aesthetics and usability suffer for the sake of improving every other element in the design.

5.2. Which platform first?

Adaptive designs can contain multiple layouts, like ones for mobile, tablet, laptop, desktop, television, etc. One of the more fundamental choices when designing these layouts is to decide which one to design first. Whichever design is worked on first will often inform the other designs, and tends to feed back into the strategy of the site, affecting its structure and content.

Currently, there are two popular choices for a first design: the narrowest design, also known as "mobile first", and the widest design, also known as "desktop first". [9.]

Today, many vocal people in the web design community vouch for the "mobile first" approach. It is thought that when designing for a large screen first, it is tempting to try to "fill up" the gratuitous amount of space, resulting in extraneous design elements being added. When designing for a small screen, more attention is paid to conserving space instead of using it, resulting in more focused and cruft-free designs. Since in an adaptive site all designs more or less share the same elements, a cruft-free mobile design also means a cruft-free desktop design. Mobile devices also tend to be less powerful in terms of performance, which should make designers and developers pay more attention to keeping sites lightweight. [9.]

In the design of New Reserve, I chose the "desktop first" approach, for two reasons. First, I was still inexperienced in designing for mobile, so starting with the desktop design seemed like a safe choice. Second, at the time I assumed the primary platform users would access New Reserve from would be PCs, thinking mobile and tablet usage would not grow fast enough to justify being made the primary target platform just yet. I am no longer sure if I made the right choice.

The growth of mobile data traffic doubled from 2010 to 2011 and shows no signs of slowing down [10]. In 2010, the amount of mobile internet usage was predicted to surpass that of the desktop internet in 2014, and every time I have seen someone try

to predict the date of this happening, it has moved closer and closer to the present day [11]. In 2010 it was also found that in countries like India and Egypt, well over 50% of users only ever use a mobile internet connection and never or seldom a desktop one, and even in the United States that number was 25% [12]. Even the amount of smartphones shipped globally already surpassed the amount of laptops and desktops in the last quarter of 2010 [9].

It seems to me that the acceleration of mobile growth is not slowing down in the least, so I would definitely recommend the "mobile first" approach to almost any website being designed today simply because most internet users will soon be browsing using mobile internet.

5.3. Using the available space

Websites are generally laid out vertically. There is no single particular reason for this. The technology around building and viewing websites just seems to have developed this way: many of the CSS features focused on laying out elements are not useful for horizontal layouts, and many of the mice people use to scroll websites today still cannot scroll horizontally in any comfortable way. The predominantly vertical nature of websites poses a problem: many design elements simply cannot be stretched beyond a certain width, meaning it is next to impossible to have a website layout take advantage of the entire screen's width on screens of all sizes.

The most popular design element on a website is often textual content, which tends to be made out of paragraphs. To ensure paragraphs are comfortable to read, it is widely recommended their width is set so that each line contains no less than 45 and no more than 75 characters. Around 66 characters is regarded as an ideal point, but it varies slightly depending on the font and line-height used. [6.]

Usually this 66 character width can be attained by setting the paragraph's width to "32em", which equals ((font-size) * 32). If we assume that the font-size being used is 16 pixels (a very common size for body text on websites), the ideal paragraph width ends up being (16 pixels * 32) = 512 pixels, which is not nearly enough to get anywhere near filling most screens in use today, which range from 180 pixels all the way up to 2560 pixels in width. Many designers try to mitigate this problem by adding "sidebars" or other auxiliary elements beside their textual content, but often that is re-

garded as a bad idea, as those additional elements can distract the user from reading the actual content of the page.

Another way to mitigate this problem is to simply increase the font-size, since a larger font-size equals a larger ideal paragraph width. Unfortunately the usefulness of this is limited, since readable body text sizes on websites are limited to around 12–20 pixels, depending on the exact pixel density of the screen being used, the distance it is being viewed from, and the user's eyesight. Even (20 pixels * 32) equals only 640 pixels; still nowhere near wide enough to fill up most screens.

Because of the reasons outlined above, the vast majority of websites that mainly include textual content end up having much empty space on their sides, especially on large screens. Websites that mainly include image or video based content can, in some cases, utilize more screen space, but they are not without problems either.

When fitting any images or videos on a screen, attention should be paid not only to their width, but also to their height; seeing an image so tall it does not fit into the browser window can be irritating. Thankfully, media queries can be used to not only detect the screen's width, but also the height. The following code example adjusts the maximum height of any images on the site to be no larger than the height of the screen. [13.]

```
@media screen and (min-height: 768px) {
      img {
            max-height: 768px;
      }
}
```

Code example 3.

Another problem with filling the screen with images or videos is the fact that they all have what is called a native resolution. If you take a 600 pixel wide image and scale it up to fill a 1024 pixel screen, the browser will have to interpolate it, making it look blurry. Images and videos can only be shown as large as their native resolutions without significant quality loss. One way to get around this is to replace the images and videos with ones that have larger or smaller native resolutions as needed. This technique is referred to as "responsive images", but more on that further on in this thesis.

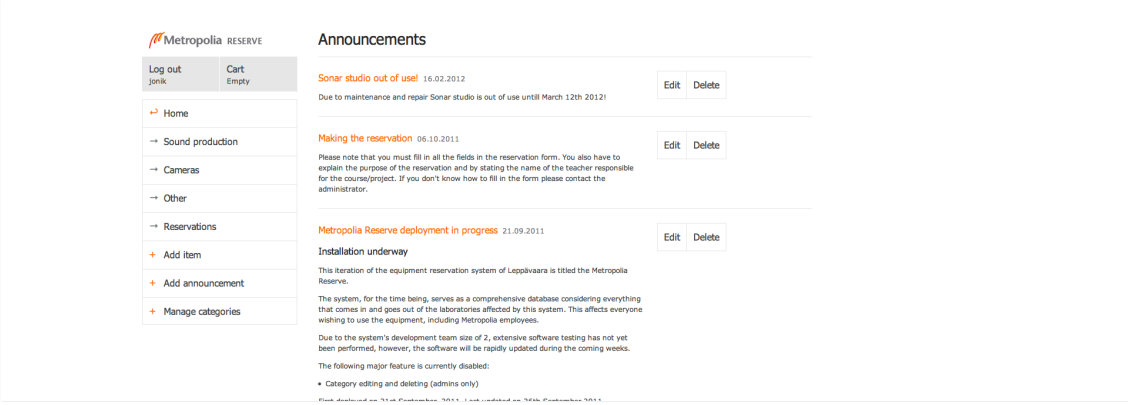

Figure 11. Plenty of empty space around the New Reserve layout on a 1920 pixel wide screen.

Because of all of the problems above, my recommendation is to simply not try to fill up every bit of horizontal space at every possible screen size, because it is nearly impossible without compromising the readability and integrity of the site's content. I followed my own recommendation in the design of New Reserve, and ended up having a whole lot of empty space on the sides at most popular screen widths.

6. Interaction design

6.1. Touch targets

Touch screens are not directly related to adaptive design, but touch-based devices became popular around the same time as it was taking off, so designing for their needs developed in tandem with adaptive design techniques, and rightfully so. I would also make the case that adapting to an input method, like a touch screen or a mouse, is just as important as adapting to a screen size or a context.

Touch screen and mouse based devices have one important thing in common: when designing user interfaces for them, it is better to make elements too big, rather than too small.

All the major touch screen operating system vendors have their own recommendations for the dimensions of touchable elements: Apple recommends 44 points (virtual pixels) square for their devices, while other vendors like Nokia and Microsoft specify their recommendations in millimetres, roughly ranging from 7–14 mm. What all of these recommendations are generally saying is that touchable elements should be roughly the

same size as an average adult fingertip or finger pad. On the web, there is currently no safe and practical way to ensure that touchable elements will always be this large, due to the unpredictability and differences of the myriad devices websites are viewed on, but a fairly safe default can be achieved by making elements at least 44 pixels square. This should, according to my limited testing, make them comfortable to touch on the majority of the different devices in use today. [14.]

Before touch screens became popular, clickable elements on websites were traditionally much smaller than 44 pixels square. I believe this had nothing to do with their usability, but rather it was done simply because the screens people used were smaller back then, so there was less space to use. According Fitts' Law, the bigger clickable elements are, the less effort it will take anyone to click them. Therefore it could be deduced that making clickable elements bigger to accommodate touch screens will not negatively affect their usability, but will actually improve it. [15.]

Even though making clickable elements big has not negative impact on usability, it does cause some problems in another realm: visual appeal. Filling up at least 44 pixels of vertical and horizontal space is not always simple, as text labels used in the average web design are rarely taller than around 16–18 pixels, which still leaves at least 26 pixels out of the 44 to fill up. Often, this can make elements like vertical link lists feel too "airy".

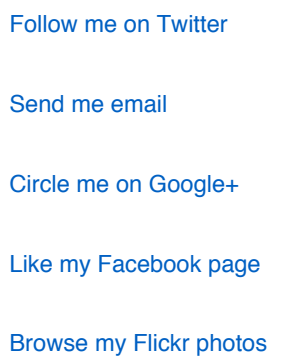

Figure 12. A lot of empty space around vertical links.

I have discovered three ways to mitigate this problem:

1. Simply make the text labels bigger. This is what is being done in the design of the Windows Phone 7 operating system.
2. Add borders around the element. This is what is being done in the design of the native elements in Apple's iOS operating system.
3. Fit more than just a single text label inside the element; icons or microcopy, for example. Both Windows Phone 7 and iOS do this to some extent.

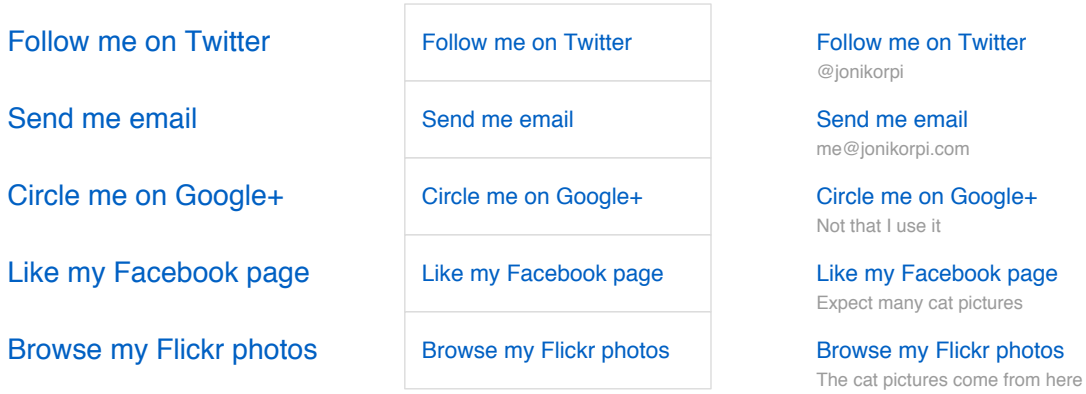

Figure 13. Different ways to deal with the extra space. Starting from the left: the Windows Phone 7 style with larger fonts, the iOS style with borders, and an example of adding more elements inside the link.

Links in the middle of paragraphs are a whole problem of their own: they absolutely cannot be made taller than whatever the paragraph's line-height happens to be, which is rarely larger than (18 * 1.5) = 27 pixels, which is much less than the recommended 44 pixels for touchable elements. I have no solution to this problem. All I can do is to try and make sure no two links in paragraphs end up too close to each other, and hope that the web browser being used is good at determining what the user is trying to touch.

Figure 14. Links in this paragraph are very close to each other. It is up to the web browser to try and figure out which one the user is tapping.

In the design of New Reserve, some touchable buttons ended up being smaller than I would have liked. The "Edit" button, for example, is only 47 pixels wide, which is not too bad in itself, since 47 pixels is still above the 44 pixel limit, but the button is also right next to the "Delete" button.

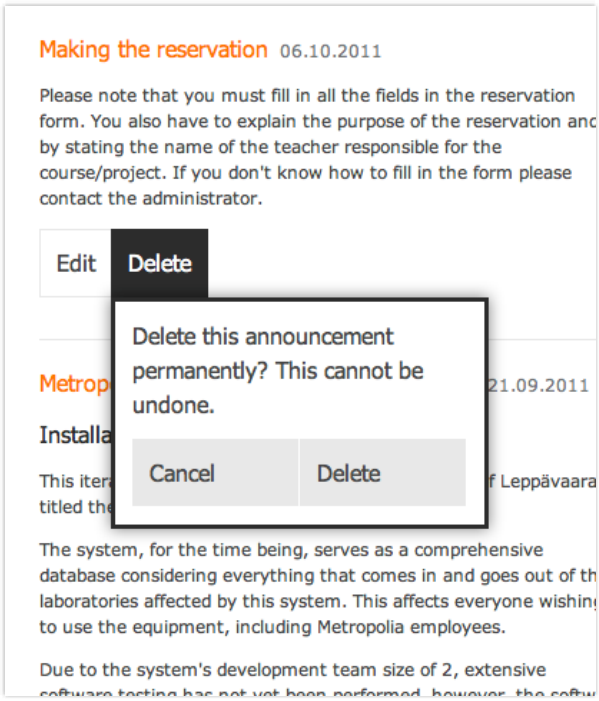

Figure 15. A confirmation dialog protects from accidental taps or clicks on buttons with destructive consequences.

It is easy to imagine how even a slight fumble might lead to the user hitting "Delete" instead of "Edit". To alleviate this problem, I did not try to artificially increase the "Edit" button's size, but instead added a confirmation dialog to the "Delete" button, as well as all other buttons that cause destructive actions.

Another type of element to be wary of in a touch-friendly design is the inline-scrollable element, meaning any element with the "overflow" attribute set to "scroll" or "auto". Since these elements take over the scrolling context, if a vertically scrollable element like this is spread to fill the entire screen, the user has to scroll to the bottom of the element before being able to scroll further down the actual page the element resides on.

6.2. Touch event delays

In most touch-based mobile web browsers today, when a user taps on a link, there is a delay of roughly 200–300 milliseconds before it fires. This delay is required because the browsers also support different kinds of touch gestures; the browser has no way of immediately knowing if the user is trying to tap a link to fire it, or just happens to be starting a swipe or a double tap gesture on top of the link, but if the browser detects no further finger movement during that 200–300 millisecond delay, it can safely assume the user was indeed tapping the link. [16.]

Unfortunately this delay is long enough to be clearly perceptible to users, easily making the website feel slow and unresponsive, which is obviously bad for usability. There are ways to circumvent the delay using Javascript. For example, instead of firing the link on the "click" event, it can be set to fire on the "touchend" event, which occurs immediately after the user lifts his or her finger off the screen. Unfortunately pulling this off in practise is not easy, as it requires a lot of fine-tuning to prevent links from accidentally firing when, for example, the user performs a double tap gesture on top of a link. [16.]

In my opinion, even a perfectly fine-tuned Javascript solution for circumventing this delay is not necessarily worth the amount of work it requires, as it is next to impossible to make it feel as responsive and familiar to the user as the touch interactions in whatever operating system and native applications they're using, which can cause a feeling of dissonance, perhaps annoying the user even more than the 200–300 millisecond delay would.

I would say this is a problem for the browser vendors to solve, and not for web developers. This is why I decided to not try to circumvent the delay in any way in New Reserve.

6.3. Animation

Animation can be an extremely powerful design tool for helping users understand and accomplish sequences of actions. Our eyes are very primed for detecting motion, so any moving element on a page will be instantly noticed, which is an especially good way to draw the user's attention to something that just occurred due to something

they just did. Today, using simple animations is possible in both modern desktop and mobile browsers, thanks to CSS transforms and transitions.

In the design of New Reserve, I originally intended to use an animation for every user-initiated action that happens inside a page — as in, anything that occurs without the page fully reloading. Due to time constraints, I only ended up applying animation to one element: the dialogs.

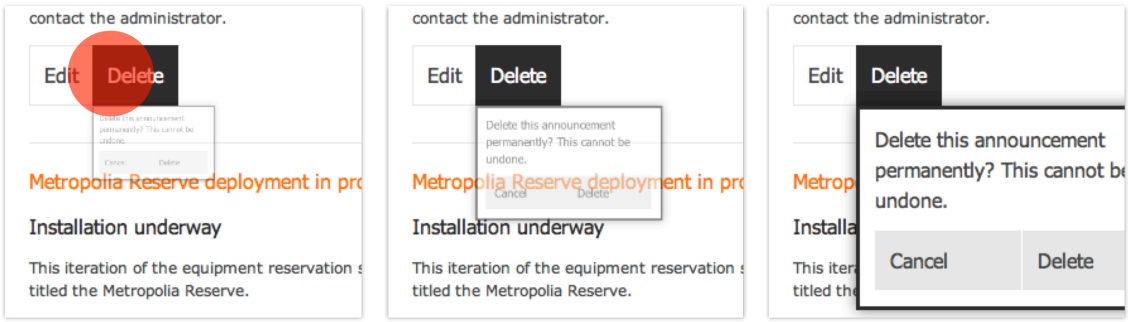

Figure 16. The dialog expanding animation.

I used a combination of a proportional scaling transition and a very quick fade-in transition to make dialog boxes spring out of whatever button caused them to appear.

```
.dialogClosed {
	transform: scaleY(0);
	opacity: 0;
	transition: all 1s ease-out;
}

.dialogOpen {
	transform: scaleY(1);
	opacity: 1;
}
```

Code example 5. A simplified version of the dialog box transition code.

I also attempted to add some animation to the site's two-level main navigation. My intention was to make the sub-menus open with a transition of some sort, instead of just letting them clumsily "blink" open. Sadly, this proved to be a difficult feat.

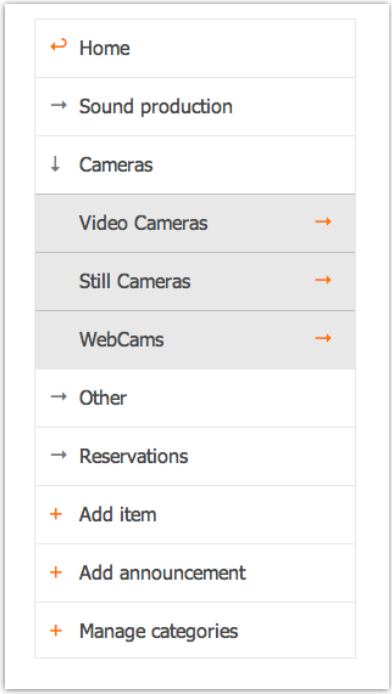

Figure 17. New Reserve's main navigation.

The problem was that whenever the height or width of an element is being transitioned, it has to transition from one defined number to another. Neither of those numbers can be set to "auto", which is the default state of dimensions in all CSS elements, basically meaning "take up as much space as required". According to my own testing, every modern browser simply ignores the transition altogether whenever an element's width or height is being transitioned from a set number to "auto".

```
.menuClosed {
      height: 0;
      transition: all 1s ease-out;
}

.menuOpen {
      height: auto;
}
```

Code example 5. This does not work in any modern browser.

The only way to get around this problem would have been to use some very inelegant Javascript. I could have, for example, tried to estimate the sub-menu's total height by counting the number of items inside it and multiplying their number by the height of a single menu item. However, there are many edge cases where this could have failed.

For example, one of the menu items could have had a particularly long label, making it wrap on two lines, which would have made it take up more vertical space than the other menu items, rendering the total height estimation useless.

Eventually due to time constraints, I gave up on trying to get the sub-menu transitions to work and left them without a transition. Later on, after the project was finished, I discovered an alternative way to create a transition similar to what animating dimensions to "auto" would have looked like. It works by using CSS scale transforms, and is actually very similar to the transition I used for the aforementioned dialog boxes.

```
.menuClosed {
      transform: scaleY(0);
      transition: all 1s ease-out;
}

.menuOpen {
      transform: scaleY(1);
}
```

Code example 6. A simplified version of the hypothetical menu transition code.

Instead of trying to animate the element's height or width, this method stretches the element from an invisibly thin line to its natural height. The result is visually slightly different from the "slide-in" effect that animating a dimension to "auto" would have looked like, but it works well enough — especially if combined with other transitions, such as a slight fade-in.

From my experiences with this project, I can say that it is good to avoid trying to animate anything by their dimensions, unless the dimensions are absolutely certainly never going to change. For elements with changing dimensions, using scale transforms combined with some other effects can stand in for dimensional transitions quite well.

7. Technical considerations

7.1. Progressive enhancement

Progressive enhancement is a development pattern very similar to the Mobile First and future-friendly approaches mentioned earlier in this thesis [4, 9]. In web design, I would describe progressive enhancement to be the act of providing rudimentary core

user interface and functionality for each of the site's features, and enriching them step by step using Javascript and modern browser features. It is a pattern that ensures that each of the site's features can be accessed and operated with almost any kind of a device and browser; users with low-powered or outdated devices or browsers will receive a rudimentary experience, while users with modern ones will receive a richer experience.

Modernizr is an extremely useful tool for developing a progressive enhanced site. It not only provides the equivalent of CSS media queries in Javascript, but also a way to test for the support of nearly every modern browser feature, and execute different batches of Javascript depending on if a feature is supported or not. [3.]

In the front-end development of New Reserve, I used Modernizr to test for HTML5 form features as follows.

```
Modernizr.load([
	rootPath + 'js/ender.min.js',
	{
		test:
			Modernizr.placeholder &&
			Modernizr.min &&
			Modernizr.max &&
			Modernizr.required &&
			Modernizr.email &&
			Modernizr.pattern
		,
		nope: {'polyForms': rootPath + 'js/polyfill-forms.js'},
		callback: {
			'polyForms': function () {
				$('form').each(function() {
					H5F.setup($(this));
					$(this).addClass('polyfilled');
				});
			}
		}
	},
	rootPath + 'js/main.js'
]);
```

Code example 7. The Modernizr load function from the New Reserve site.

First of all, the function above asynchronously loads the Ender Javascript library, which is used across the site [17]. The function then proceeds to perform a series of tests to determine whether the browser being used supports all of the modern HTML5 form features being used in New Reserve. If any of them are unsupported, a HTML5 form polyfill is asynchronously loaded and consequently applied to any forms present on the page. If all the features are supported, however, the polyfill does not get loaded, thus saving a considerable amount of loading time and resources, making the page load faster.

Progressively enhanced features, like the HTML5 form ones above, can be combined with Modernizr's match media queries, allowing different polyfills and batches of Javascript to be executed depending on, for example, the dimensions of the browser or device being used to access the site [3]. In New Reserve, I ended up not having to do this, since CSS did most of the work.

7.2. Restrictive back-ends and impossible polyfills

When picking a content management system or a development framework to build an adaptive website on, it is best to make sure that using it to implement modern HTML5 features — such as HTML5 form elements — is possible. This is because building a user interface that adapts to both touch and mouse devices often requires these modern features.

In the design of New Reserve, I needed to often use form elements to let users input dates, which can be cumbersome to type out, especially using small touch screen keyboards. I wanted to use the new date and time input elements included in HTML5, because many touch screen devices adapt to these elements by providing a special native interface, which makes for a much better user experience than a regular keyboard input interface. Sadly, the development framework we were building New Reserve on, CakePHP, did not allow us to use these new elements because of a technical quirk.

This was not the only problem I faced in attempting to use the new HTML5 form elements. Even though they degrade gracefully, turning into regular text input elements in browsers that do no expressly support them, actually browser support for their new interfaces is still relatively scarce [18]. These elements are also extremely difficult to

"polyfill" (meaning the act of using Javascript to add support for a feature in browsers that do not support it), because the interfaces they provide vary wildly between different types of devices, meaning the polyfill would most likely need to provide several different interfaces and serve them appropriately to different devices.

In the end, we opted to use a series of <select> elements for selecting dates and times: one element for hours, one for minutes, one for months, and so on. The interfaces they provide are not quite as elegant as they could be, but they work reasonably well on a wide array of devices, providing a better interface for selecting dates and times than simple text fields do.

Form validation is another important HTML5 feature that is both difficult to polyfill and can be poorly supported in some content management systems and frameworks [18]. I have found that it is also an extremely important feature to get right across all devices, because as painful as having a badly designed form error out on you can be, I have found it to be endlessly worse if it happens on a device where text input is taxing, such as a small touch screen device. It is not hard to imagine users abandoning a form they were attempting to complete if it errors out in a confusing way and forces them to re-input a large amount of text. This can be especially damaging to the kind of websites where filling forms is required for the user to accomplish tasks or for the user to be considered a "conversion" (meaning the website has succeeded in getting them to accomplish a goal beneficial to the website).

7.3. Issues with the box model

The CSS "box model" causes a lot of technical issues in adaptive design and development, especially when a fluid grid is being used. The box model defines the behaviour of dimensions, such as widths, paddings, margins and borders given to an HTML element. The total dimensions of an element are calculated by adding up its width or height and any margins, paddings or borders it might have. This behaviour is best explained with an illustration.

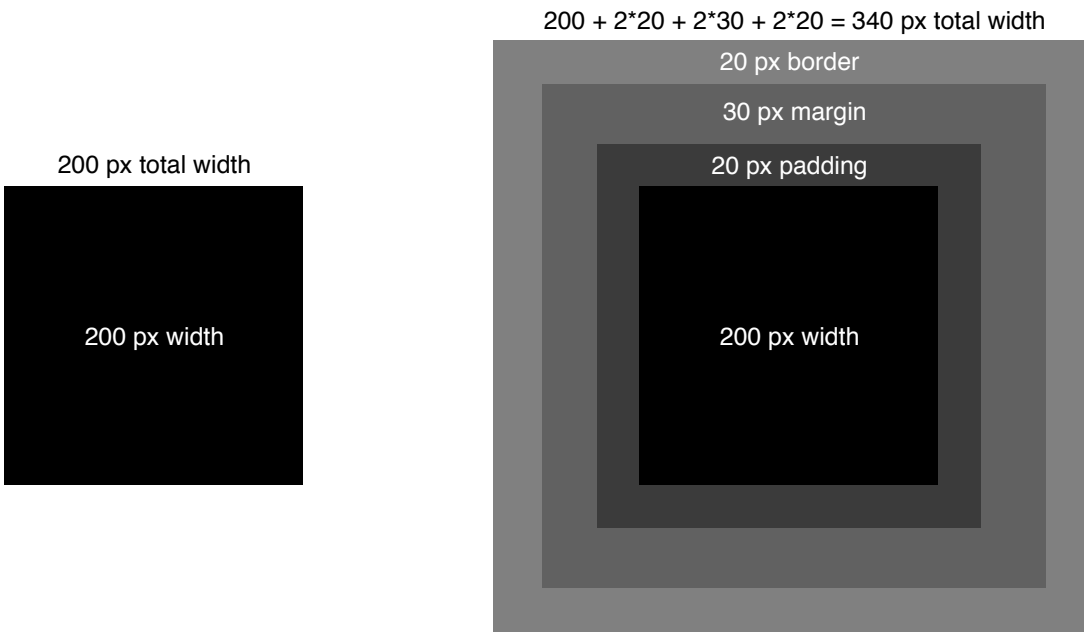

Figure 18. An illustration of how the default CSS box model behaves.

An element that has been given a width of 200 pixels is — as expected — 200 pixels wide. When the same element is given 20 pixels of padding, 20 pixels of margins and a 10 pixel wide border, its total width is suddenly 250 pixels, even though its width is still set to 200 pixels.

This behaviour is problematic not only because it requires the coder to constantly manually subtract the paddings, margins and borders of each element from its width or height to avoid making its dimensions too large by accident, but also because when using a fluid grid, elements often require using a mix of percentage units and em or pixel units in their dimensions. For example, a paragraph's width might be set to 100%, while its padding might be set to 20 pixels, making its total width (100% of its containing element's width) + 2 * (20 pixels).

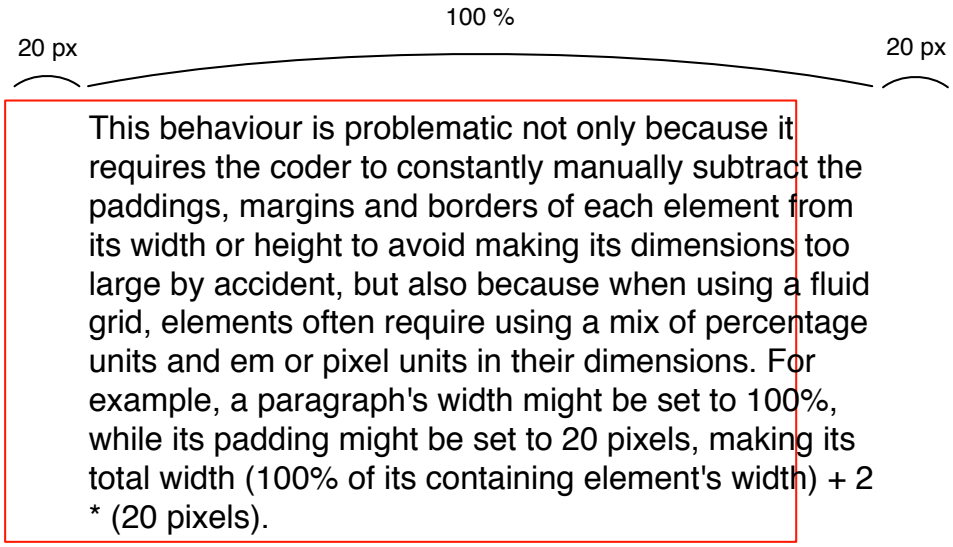

Figure 19. A paragraph overflowing out of its containing element (shown in red).

Thankfully, most of the current modern web browsers support an alternative, more intuitive box-model called "border-box". Its behaviour is once again best explained with an illustration. [19.]

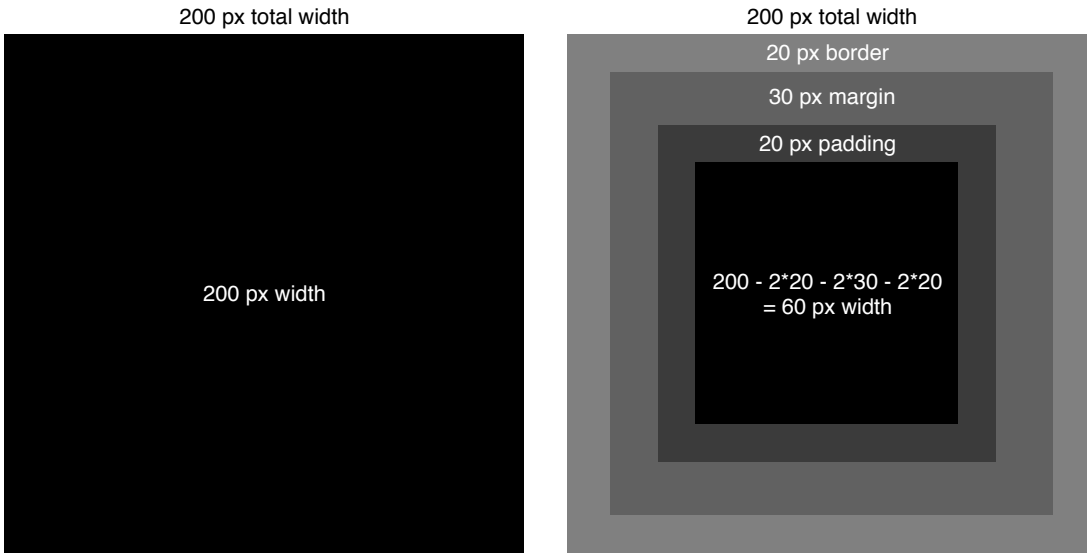

Figure 20. An illustration of the more intuitive and less troublesome "border-box" model.

In an element using the border-box model, paddings, margins, and borders are no longer added to the element's total dimensions, but subtracted from it. This makes

coding simpler and sidesteps problems caused by mixing percentage units and absolute units. Enabling the border-box model is fairly simple: the universal "*" selector can be used to select every element on the page. [19.]

```
* {
      -webkit-box-sizing: border-box;
      -moz-box-sizing: border-box;
      box-sizing: border-box;
}
```

Code example 7. A simple way to set every element on the page to use the border-box model [19].

Sadly, the border-box model is only supported in fairly modern browsers: Internet Explorer 7 and its ancestors will not even recognize the "box-sizing" CSS attribute [19]. At the time of writing, it is unknown whether non-modern mobile browsers support it, since CSS feature support has not been thoroughly researched on those kinds devices. Therefore I would personally advice against using the border-box model without some sort of a surefire way to make sure it behaves as expected in non-modern desktop and mobile browsers as well.

In the design of New Reserve, I ran into box-model-related trouble when setting the dimensions of form elements. Since their cannot be set to "auto" — which would make them automatically fill up all the space there is available — I tried to set their widths to "100%", which caused them to flow out of their containing elements, because each form element also has some margins, paddings and a border. I could not use the border-box model due to the aforementioned browser support issues, so I sidestepped the problem by setting their widths in absolute units in each important threshold of the adaptive layout: 320, 480 and 768 pixels. As a result, they only flow out of their containing elements at screen widths lower than 320 pixels, and end up not filling up all the available space in screen widths that end up between the thresholds. It is not a perfect solution by any means, but it makes the form elements behave in a visually pleasing way in the majority of use cases.

7.4. Fluid media and responsive images

Fluid media is the third component of responsive web design, as defined by Ethan Marcotte [1]. It means making any images and videos on a page proportionally scale up or down to fill whatever space is available — at least until their native resolutions are

reached and scaling them up would deteriorate their visual quality. This is an easy feat to implement technically, but when bandwidth and file size restrictions are taken into account, problems arise.

If the page is viewed on a 320 pixel wide screen, should it be served a fluid image or video with a native width of 1280 pixels? How about when the screen is 320 _virtual_ pixels wide, which are comprised out of 640 physical pixels? Solving problems like these requires the ability to serve images of different sizes for different devices, an ability that does not currently exist in any elegant form for images, but does for videos. This problem is currently being worked on under the term "responsive images".

Videos are easy to make "responsive". The HTML5 video specification allows for multiple "source" elements, which can be set to be only served to appropriate devices and browsers [20]. To my knowledge, doing something similar with Flash or Silverlight-based video should also be possible.

There are currently many techniques for making images responsive in the same way as videos, but none of them are elegant, gracefully degrading and easy to implement all at the same time [21]. During the writing of this thesis, a W3C working group for responsive images was established, and is working on both bringing native responsive image support to modern browsers and creating and accompanying polyfill. [22.]

In the development of New Reserve, I opted not to use any techniques for making images on the site responsive, because images were not and important element in the functionality of the site. No support for responsive videos was added either, since the administrators were advised to make textual links to any video content they might want to reference on the site, instead of directly embedding them.

8. Style guides

Using style guides or brand guidelines in adaptive web design and modern web design in general should be done with care. If they were written in a time before adaptive web design became popular, the guidelines will most likely not take into account the inherent unpredictability of the myriad screen sizes and input methods in use today and will lack the flexibility required to design for them.

In the design of New Reserve, I attempted to use the Metropolia style guide and UI standards. They were obviously not written with touch interfaces, mobile operating systems, or varying screen sizes in mind, so I had to ignore some parts guidelines and make new ones up as I went along.

One such instance was when I was designing the dialog boxes used in confirming actions. According to the Metropolia UI standards, the "Okay" buttons in dialog boxes should always go on the left and "Cancel" buttons should go on the right, but buttons signifying moving onwards, such as "Next", should also go on the right [23]. This is, I believe, an UI tradition derived from Windows, one which does not correspond to how most modern operating systems work today.

According to usability researcher Jakob Nielsen, placing the "Okay" button on the right can improve the "flow" of using the dialog box for a western user, as we scan the buttons from left to right before deciding which one of them to select [24]. Assuming we want to select the "Okay" button, our eyes have to first scan the row of buttons from left to right, then jump back towards the left to seek out the "Okay" button again to select it. If the "Okay" button is on the right, it is where our eyes will naturally stop after the initial scan.

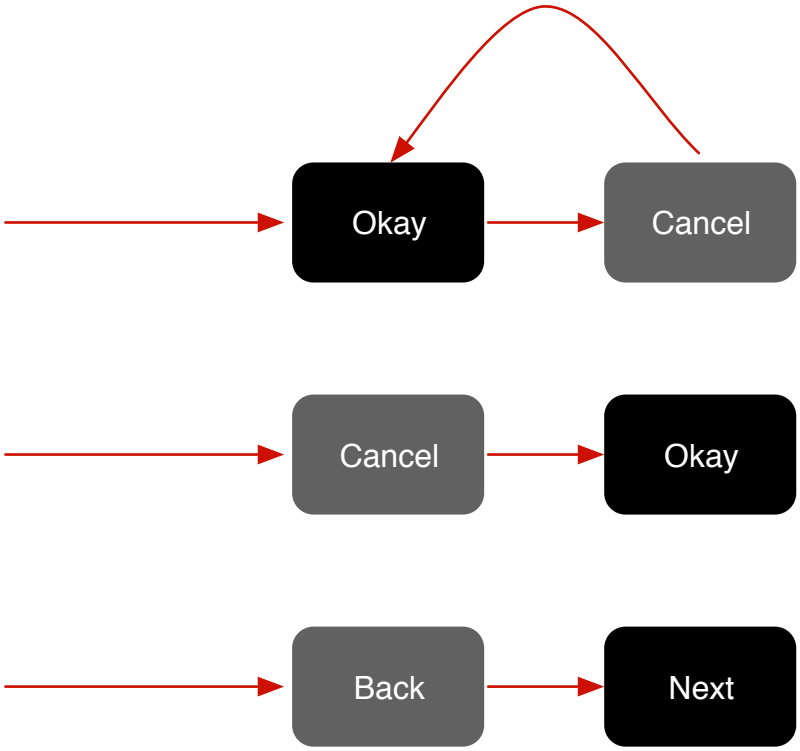

Figure 21. An illustration of how western users are likely to scan rows of buttons, assuming they are looking for the "Okay" or "Next" buttons (shown in a darker colour).

To me, "Okay" and "Next" are both actions that associated with taking a step forward, while "Cancel" and "Back" are associated with taking a step backward. It seems logical that the placement of the buttons should be determined by whether they make the user take a step forward or backward. According to my observations, "Okay" is only ever placed on the left in Windows. In every other popular desktop and mobile operating system I have used, "Okay" is placed on the right. In the design of New Reserve, I placed "Okay" on the right.

Another instance of guidelines colliding with modern requirements occurred when I attempted to design New Reserve's typographic elements to be consistent with the Metropolia style guide. It recommended the use of Verdana for body text. I would say Verdana is only readable in fairly small sizes, and modern websites generally require a font that is also readable in larger sizes, since it is often beneficial for text to take up larger amounts of space on large screens. In this case, I ended up simply always keeping body text small, so there was no need to stray away from the guideline. [23.]

9. Conclusions

Using adaptive web design techniques in web applications is difficult, but not in any way impossible to pull off. The biggest pain points in the process are keeping visual design appealing and trying to support multiple input methods at the same time.

Designing with a fixed-width grid may be a good idea for designers inexperienced with adaptive design, as it makes empty space and element dimensions easier to manage. When using a fluid grid, enabling the "border-box" model can help a great deal when dealing with elements the dimensions of which are specified in both percentages and pixels or ems.

Style guides written before the advent of adaptive design or written by designers not acquainted with it should be used with caution, as many guidelines may be unsuitable for use in designs for small screens or large modern screens.

When choosing a design to start out from, choosing the "Mobile First" approach is a good idea for almost any website being developed today. It will result in more cruft-free designs and prepare the site to better serve the immensely quickly growing amount of mobile internet users. In wider designs, I would recommend not trying to fill up all available horizontal space on the screen, as it would not necessarily benefit the content of the site in any way.

Touch screens should be kept in mind when designing adaptive websites. Making all touchable elements at least 44 pixels square is a good rule of thumb. If it cannot be adhered to, it is best to try and mitigate the possible negative impacts of such a small touch target in ways other than attempting to change its dimensions.

When animating UI elements that may have unpredictable dimensions, it is best to use scaling transforms and other effects instead of attempting to animate the element's dimensions directly to avoid technical issues.

Making images adaptive in terms of resolution and bandwidth usage is difficult at the moment, as there are many techniques to choose from, but none of them are very elegant. However, a more standards-based solution may be just around the corner. Videos can already be made responsive using the HTML5 video element.

Progressive enhancement is the key in building an adaptive web application. Modernizr is a very useful tool for taking advantage of modern browser features without leaving older browsers behind. However, when using it to add functionality to older browsers using polyfills, it is good to exercise caution, since devices with different input methods may require different kinds of polyfills.

References

1. Marcotte, Ethan. 2011. Responsive Web Design. Electronic book. A Book Apart.

2. Media Queries. W3C. Web document. <http://www.w3.org/TR/css3-mediaqueries/>. 27.7.2010. Read on 3.2.2012.

3. Ateş, Faruk; Irish, Paul; Sexton, Alex. Modernizr. Web document. <http://www.modernizr.com/docs/>. Read on 16.1.2012.

4. Wroblewski, Luke; Jenson, Scott; Frost, Brad; Keith, Jeremy; Gardner, Lyza D.; Jehl, Scott; Rieger, Stephanie; Grigsby, Jason; Rieger, Bryan; Clark, Josh. 2011. Future Friendly. Web document. <http://futurefriend.ly/thinking.html>. 2011. Read on 23.1.2012.

5. Marcotte, Ethan. 2009. Fluid Grids. Web document. <http://www.alistapart.com/articles/fluidgrids/>. 3.3.2009. Read on 16.1.2012.

6. Rutter, Richard. 2010. Choose a comfortable measure. Web document. <http://webtypography.net/Rhythm_and_Proportion/Horizontal_Motion/2.1.2/>. 2010. Read on 11.8.2011.

7. Knight, Kayla. 2009. Fixed vs. Fluid vs. Elastic Layout: What's The Right One For You? Web document. <http://coding.smashingmagazine.com/2009/06/02/fixed-vs-fluid-vs-elastic-layout-whats-the-right-one-for-you/>. 2.6.2009. Read on 16.1.2012.

8. Korpi, Joni. 2011. Golden Grid System. Web document. <http://goldengridsystem.com>. 2011. Read on 8.2.2012.

9. Wroblewski, Luke. 2011. Mobile First. Electronic book. A Book Apart.

10. Mobile data traffic growth doubled over one year. 2011. Ericsson. Web document. <http://www.ericsson.com/news/111012_mobile_data_traffic_244188808_c>. 12.10.2011. Read on 12.2.2012.

11. Meeker, Mary; Devitt, Scott; Wu, Liang. 2010. Internet Trends. Morgan Stanley. Web document. <http://www.morganstanley.com/institutional/techresearch/pdfs/Internet_Trends_041210.pdf>. 12.4.2010. Read on 12.2.2012.

12. The Mobile Only Internet Generation. 2010. On Device Research. Web document. <http://www.slideshare.net/OnDevice/the-mobile-only-internet-generation>. 14.12.2010. Read on 12.2.2012.

13. Walton, Trent. 2012. Vertical Media Queries & Wide Sites. Web document. <http://trentwalton.com/2012/01/11/vertical-media-queries-wide-sites/>. 11.1.2012. Read on 16.1.2012.

14. Wroblewski, Luke. 2010. Touch Target Sizes. <http://www.lukew.com/ff/entry.asp?1085>. 4.5.2010. Read on 18.2.2012.

15. Gross, Jason. 2011. Improving Usability with Fitts' Law. <http://sixrevisions.com/usabilityaccessibility/improving-usability-with-fitts-law/>. 17.5.2011. Read on 18.2.2012.

16. Fioravanti, Ryan. 2011. Creating Fast Buttons for Mobile Web Applications. Google. <http://code.google.com/mobile/articles/fast_buttons.html>. 1/2011. Read on 21.2.2012.

17. Diaz, Dustin & Thornton, Jacob. 2012. Ender. Web document. <http://ender.no.de/>. 2012. Read on 14.3.2012.

18. Deveria, Alexis. 2012. When can I use... Compatibility tables for support of HTML5, CSS3, SVG. Web document. <http://caniuse.com/>. 5.3.2012. Read on 14.3.2012.

19. Irish, Paul. 2012. * { box-sizing: border-box } FTW. Web document. <http://paulirish.com/2012/box-sizing-border-box-ftw/>. 1.2.2012. Read on 2.2.2012.

20. Ernest Delgado. 2010. HTML5 Video. Web document. <http://www.html5rocks.com/en/tutorials/video/basics/>. 3.8.2010. Read on 14.3.2012.

21. Grigsby, Jason. 2011. Preferred solutions for responsive images. Web document. <http://www.cloudfour.com/preferred-solutions-for-responsive-images/>. 22.11.2011. Read on 20.1.2012.

22. Marquis, Mathew. 2012. Responsive Images Community Group. Web document. <http://www.w3.org/community/respimg/>. 2012. Read on 14.3.2012.

23. Kaipila, Matti. 2010. Metropolian käyttöliittymästandardi. Web document. <http://www.metropolia.fi/fileadmin/template/aineistopankki/materiaali/graafinen_ohjeisto/kayttoliittymastandardi/Metropolian_UI_10.pdf>. 24.6.2010. Read on 16.1.2012.

24. Nielsen, Jakob. 2008. OK–Cancel or Cancel–OK? Web document. Alertbox. <http://www.useit.com/alertbox/ok-cancel.html>. 27.5.2008. Read on 4.8.2011.