Maxim Dolgobrod

Developing a user interface for a cross-platform web application

Helsinki Metropolia University of Applied Sciences
Master's Thesis
Information Technology
30 May 2013

Metropolia

Abstract

Author(s) Title Number of Pages Date	Maxim Dolgobrod Developing a user interface for a cross-platform web application 117 pages 30 May 2013
Degree	Master of Engineering
Degree Programme	Information Technology
Specialisation option	Media Engineering
Supervisor	Aarne Klemetti, Senior Lecturer

The purpose of this master thesis project was to investigate and analyse the main design and development approaches to creating a user interface of a cross-platform web application that is optimised for usage on both mobile and non-mobile devices. The additional goals were to analyse the main challenges in implementing such a user interface and find out whether it is feasible to achieve a consistent user experience both on mobile and desktop devices.

The theoretical part of this paper analyses the main development approaches and design considerations for creating a user interface that works on mobile and non-mobile devices. For the practical part, a prototype user interface for a cross-platform book cataloguing web application has been built, tested on a number of mobile and non-mobile devices and evaluated in terms of performance and user experience. Also ideas for its further development are presented.

As the result of this project, a semi-functional web application prototype has been built using Backbone.js and user interface has been created in HTML5 with CSS3. The application allows the user to organise his/her books and planned reads, leave reviews and view reviews from other readers.

The application did not have all the functioning features and it still needs further work before actual usage. The biggest challenges were content planning, designing reusable UI components, finding a suitable framework for the application implementation and inconsistent CSS support across browsers. By planning the navigation and content early, testing the user interface on real devices, following the mobile first approach and progressively enhancing to a device's capabilities, it is possible to create a consistent user experience across mobile and non-mobile devices in lightweight web applications.

Keywords	Cross-platform UI, cross-platform web application, mobile web, responsive web design, HTML5

Contents

Abbreviations and terms

3G	3rd Generation Mobile Telecommunications
AMD	Asynchronous Module Definition
AJAX	Asynchronous JavaScript And XML
API	Application Programming Interface
CDN	Content Delivery Network
CSS	Cascading Style Sheets
CSS3	Cascading Style Sheets level 3
DOM	Document Object Model
GUI	Graphical User Interface
JSON	JavaScript Object Notation
HTML	HyperText Markup Language
HTML5	HyperText Markup Language version 5
HTTP	HyperText Transfer Protocol
ODP	On-Device Portal
OS	Operating System
RWD	Responsive Web Design
RESS	Responsive Web Design and Server Side components
REST	Representational State Transfer
SASS	Syntatically Awesome Stylesheets
SVG	Scalable Vector Graphics
SQL	Structured Query Language
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WHATWG	Web Hypertext Application Technology Working Group
WiFi	Wireless Local Area Network
WML	Wireless Markup Language
WWW	World Wide Web
XHTML	eXtensible Hypertext Markup Language
XHTML MP	eXtensible Hypertext Markup Language Mobile Profile
XML	Extensible Markup Language
UA	User-Agent
UI	User Interface
URL	Uniform Resource Locator
UX	User Experience

1 Introduction

Mobile services nowadays are available in the form of mobile applications or web services that can be accessed through the web browser. With a rapidly growing amount of mobile devices, which increased from 490 million units in 2011 to 700 units in 2012, the mobile market is becoming increasingly attractive for content and service providers. [1]

The most common approach for providing services to mobile users is to create a mobile website or develop a native mobile application, which runs only on the platform for which it was designed. For the biggest coverage of audience in these cases there would then be a separate application for iOS, Android, Windows Phone, BlackBerry and Web OS devices. Developing a native application guarantees that users have the best possible user experience on their devices. [2, 3]

Choosing what mobile platforms to support continues to be a puzzling problem for developers both big and small. Supporting one platform can be difficult enough, but nowadays developers not only have multiple operating systems to consider, but multiple device types as well. Therefore, the growing amount of various mobile devices also makes it more challenging to deliver the content while also maintaining the user experience at a high level.

In some cases it might not always be feasible to have a separate application for each platform and device. The solution in this case would be to create a mobile web application that is optimised for mobile devices. [2, 4] The added benefit of this approach is that the same backend can be used to run the application also for desktop computers. However, a cross-platform web application should be able to adapt to different screen sizes, orientations, device capabilities and varying interaction methods in order provide an adequate user experience across various devices. [3] While there are a number of solutions for creating mobile cross-platform web apps, developing an adaptive user interface that works both for mobile devices and desktop computers can be challenging.

In this master thesis project I present a proof-of-concept design and implementation of a user interface for a cross-platform web application. By a cross-platform web

application I mean an application that can be accessed on smartphones, tablets and non-portable computers using a web browser. This goal of this project is to investigate and analyse main development and design approaches for creating the user interface of a cross-platform application. As a case study I have developed a prototype interface for a book cataloging application that allows the user to organize his/her books and planned reads, leave reviews and view reviews from other readers. The prototype provides only limited functionality and has been built purely for research purposes.

I have used the prototype to test a cross-platform interface on a number of mobile and non-mobile devices and answer the following research questions set for this project:

- What are the main challenges in developing a cross-platform user interface without using any cross-platform UI web frameworks?
- What are the most important design considerations when developing a cross-platform UI?
- Is it feasible to achieve a consistent user experience both on mobile and desktop devices in a cross-platform web application?

The project was carried out as a personal research project and was not done for a company or an actual client.

2 Mobile Ecosystem Fragmentation

This chapter describes the development of the mobile web, provides a categorisation of mobile devices, describes their hardware diversity and presents different types of mobile applications.

2.1 Evolution of the Mobile Web

WAP 1.0

The first mobile web was by introduced in early 2000 and was defined by the WAP 1.0 standard. WAP, which stands for Wireless Application Protocol, recommended using Wireless Markup Language (WML) for content creation. [4, 96] WML is an XML version designed for mobile devices that was not compatible with HTML standards. Sites developed using the WAP standard were thus referred to as "WAP sites" instead of "websites". This brought a clear separation between the WAP environment and the Web.

In order to browse WAP content mobile devices had to use WAP enabled browsers. At that time mobile devices had black and white screens with limited image support and could display only a small amount of text. Browsing at that time was also expensive, since the connection was using a voice call as a modem communication and every minute spent browsing was charged for as a voice call minute. Very few useful services were available in that version of the mobile Internet. [4, 55] Most modern devices nowadays do not support WML content anymore.

WAP 2.0

Starting from 2001, cellular networks have introduced General Packet Radio Service (GPRS) technology into their infrastructure. The initial data-transmission speed ran to around 28 kilobytes per second, but eventually GPRS phones could surf the Web at 60 kilobytes per second. Data packaging made GPRS cost-efficient, as phone users only paid for the amount of transferred data rather than for time-based usage. [4, 55] A year later WAP 2.0 was released and has remained in use to this day. This standard is

significantly closer to web standards and allows HTTP communication between device and server.

Instead of using the deprecated WML, XHTML MP (Mobile Profile) was recommended for designing the content for WAP 2.0. XHTML Mobile Profile is a subset of XHTML, which is the stricter version of HTML, with some additional elements and attributes from the full version of XHTML. [5] The primary goal of XHTML Mobile Profile is to bring together the technologies for mobile web browsing. With XHTML MP developers can use HTML/XHTML to organize the content and CSS for styling.

As a result of limitations in the multimedia features in WAP 2.0, many medium-sized and large websites started to create applications to be installed on the users' mobile devices. Figure 1 shows "Yahoo! Go", one of the first on-device portal applications created by Yahoo. [4, 58] This application offered a better experience for the mobile users on different devices as compared to WML sites and it could be easily accessed from the mobile device's application menu. However, one significant disadvantage of this approach was that a different version of the application had to be created for each device platform.

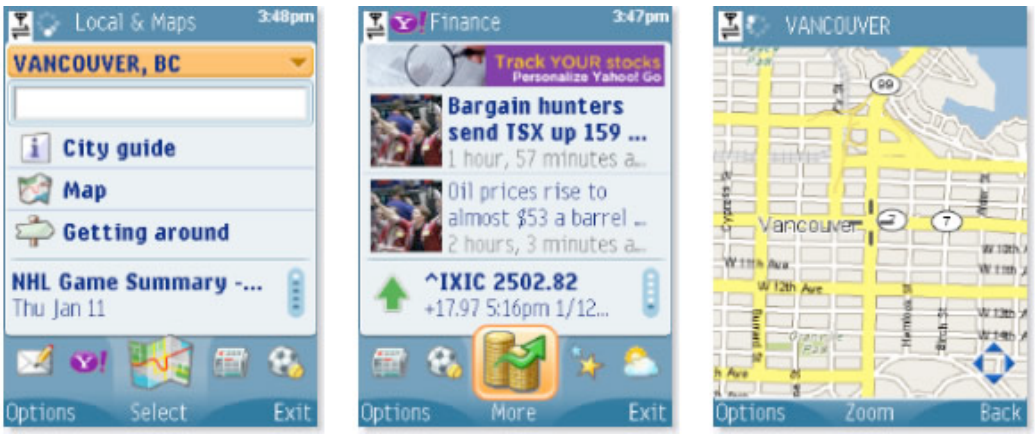


Figure 1. Yahoo! Go 2.0 version of the portal on Nokia S60 device. Copied from [6].

Mobile Web 2.0

Year 2007 saw another milestone in the development of the mobile web as a new kind of smartphones started to appear, such as the iPhone and Nokia N95 as well as Android devices. [4, 59] The significant difference to the devices previously in use was

that these new smartphones had much more advanced features for mobile browsing: bigger screens and faster connectivity with 3G and WiFi support as well as browsers capable of displaying full desktop versions of websites with AJAX and streaming video. All this made for a much richer and more diverse user experience on the mobile web and eventually led to the appearance of Mobile Web 2.0. [4, 59] Since then, devices have continued to improve in terms of connectivity, processing power and web standards support.

2.2 Classification of Mobile Devices

Firtman defines a "mobile device" as a device that is portable, personal, almost always with you, easy and fast to use as well as offering some kind of network connectivity. [4, 4] However, nowadays the personal factor is not necessarily a defining criterion anymore. For example, it is common to share e.g. an Apple iPad within the same household, whereas smartphones are likely to remain for solely personal use. [4, 5; 7]

Netbooks or laptops are also portable and may have constant network connectivity even outside the home or office, but does this make these devices mobile devices too? According to Firtman's definition, a mobile device has to be easy and fast to use, meaning that you can use it even while walking. [4, 5] Normally with a netbook or a laptop you cannot use it while moving or at least doing so won't be easy. In order for a device to classify as a mobile device it has to be portable enough to be used almost anywhere. Thus, netbooks and laptops do not qualify.

Mobile devices can be ordered into different groups based on their technical capabilities and features. In this paper I will use Firtman's classification, which is based on mobile web support and groups devices as described below.

Mobile Phones

Mobile phones are devices with very basic capabilities. They can make phone calls and send SMS messages. They usually do not have any connectivity to the Internet and therefore no web browsing is possible. Also, no application installations are typically possible, at least by the user himself. [4, 6]

Low-end Mobile Devices

Low-end mobile devices have small screens, network connectivity support and a basic web browser. These devices do not have a touch screens, come with a very limited amount of memory and might include a very basic camera. [4, 6]

Mid-end Mobile Devices

Mid-end end mobile devices provide better mobile Web browsing. They balance between an affordable price and a good user experience with medium-size screens, basic HTML-browser support, 3G, a decent camera and application support. One key aspect of these devices is that they run a not well-known proprietary operating system without any portability across manufacturers. [4, 7] These devices are also referred to as feature phones.

High-end Mobile Devices

High-end devices may have a touch screen but do not have multitouch support. However, while they do have advanced features such as an accelerometer, a decent camera, Bluetooth and good mobile browsing support, they do not provide the improved user experience available in smartphones. [4, 7]

Smartphones

It is rather difficult to have a clear definition for the mobile devices that could be classified as a smartphone, since every year high-end and mid-end devices are getting new technical improvements. Typically, though, smartphones have a multitasking operating system, a browser capable of desktop browsing, WLAN and 3G connectivity, Bluetooth, multitouch support, GPS navigation, an accelerometer and a number of other features. [4, 8] Devices in this category include Apple iPhones and Android phones as well as some Symbian phones such as Nokia N8, Windows phones Lumia 800 and Lumia 710.

Non-phone Devices

There are also mobiles devices that have the capabilities of a smartphone device, but that lack the voice support necessary for using mobile operator services, i.e. making phone calls over a GSM-network unavailable. An example of such a device would be the Apple iPod Touch. It is a very portable and easy-to-use device, has WLAN support and the same mobile browser as an iPhone, but lacks mobile network connectivity.

Some eBook readers also have Internet connectivity and a browser. For example, the Amazon Kindle 2 is able to show very basic Web pages primarily with text only content. However, browsers on eBook devices are very limited in terms of web standards support and therefore eBook readers are not included in the analysis in this project.

Tablets

A tablet is a mobile computer, bigger than a mobile phone, integrated into a flat touch screen and primarily operated by touching the screen rather than using a physical keyboard. [8] Tablets come with WLAN support, usually also with 3G connectivity, with GPS navigation, and lately also with video camera functions. The most well known examples of tablets are Apple's iPad and Samsung's Galaxy Tab. The third generation iPad has a screen with 2048 by 1536 pixels and a resolution with 3.1 million pixels, which is much larger than the most popular screen resolution, 1366 by 768 pixels, for non-mobile computers. [9; 10]

In 2011 smartphones surpassed low-end phones as the most purchased device type in Germany, France, Italy, Canada and the U.S. and the same was estimated to happen in the UK and Spain by 2012. [11] One of the top considerations in purchasing smartphones, besides the network quality of the mobile service provider, was the smartphone's operating system and the selection of apps available for that particular smartphone platform. [11] Smartphones are most popular among consumers in the age groups between 25 and 34 years old, followed by the 35-44 year olds.

2.3 Mobile Platforms

There are at least six leading mobile platforms. Among these, the Android, iOS and Windows Phone operating systems have gained the largest market shares (appendix 2, figure 2). Each of these platforms has own architecture, UI guidelines and development infrastructure.

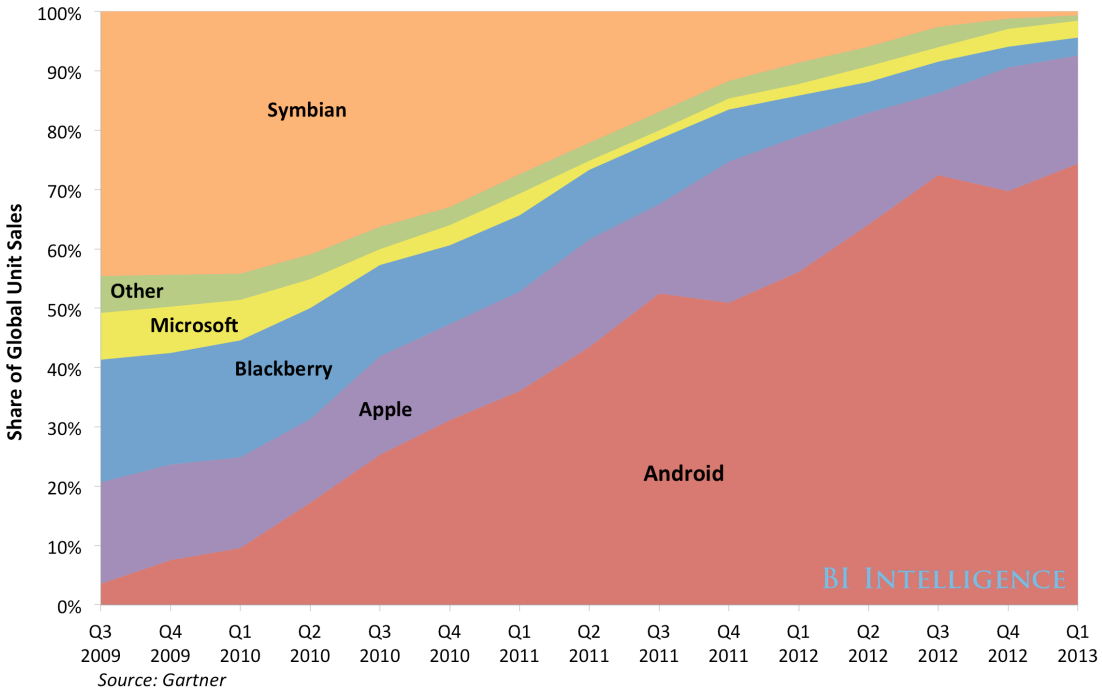


Figure 2. Global smartphone market share by platform. Copied from [12].

Android

Android is the currently leading smartphone operating system maintained by Google. [12]. Android is run on almost 4000 different mobile device models. [13] Table 1 shows the distribution of screen sizes and the densities of Android devices that accessed the Android App Store during a seven-day-period in 2012. Based on this data, a large share of the devices in use, have a normal-sized screen with high and extra density.

	ldpi	mdpi	hdpi	xhdpi
Small	1.7%		1.0%	
Normal	0.4%	11%	50.1%	25.1%
Large	0.1%	2.4%		3.6%
X-large		4.6%		

Table 1. Screen sizes and densities of Android devices that access Google Play store during 7-day period in October 2012. Data gathered from [14; 15].

Screen density means the quantity of pixels within a physical area of the screen. It is usually referred to as dots per inch (DPI). As an example, a ”low” density screen has fewer pixels in the same physical area than ”normal” or ”high” density screen. [16]

Android devices support a large number of screen sizes and densities; therefore to simplify user interface design for various screen configurations, all screen sizes and densities are divided into four generalized groups as shown in table 2.

Generalized screen size	Generalized density	Actual size (inches)	Actual density (dp)
small	ldpi	~ 2–3.5	426 x 320
normal	mdpi	~ 3.5–4	470 x 320
large	hdpi	~ 4–7	640 x 480
xlarge	xhdpi	~ 7–10	960 x 720

Table 2. Rough mapping of actual screen sizes and densities to generalised sizes and densities. Data gathered from [14; 15].

Android uses a WebKit based browser, an open-source rendering engine developed by Apple [17, 256].

Apple iOS

iOS is the second largest smartphone operating system based on the current market share. [13] iOS is run on a limited amount of devices and these devices have much less variation in screen size and density as compared to Android devices (table 3).

	Screen resolution (px)	Screen density (ppi)
iPhone 1, 3G, 3GS, iPod Touch 1–3	320 x 480	163
iPhone 4, iPhone 4S, iPod Touch 4g	960 x 640	326
iPod Touch 5g, iPhone 5	1136 x 64	326
iPad, iPad2	1024 × 768	132
iPad Mini	1024 × 768	163
iPad (3rd generation)	2048 × 1536	264

Table 3. Screen sizes and densities of iOS devices. Data gathered from [18].

Apple measures screen density in pixels per inch (PPI) instead of dpi. The default browser on iOS devices is Safari, which is based on the same WebKit engine as Android's browser.

Windows Phone

Based on the statistics of the second quarter in 2012, the Windows Phone occupies only the fifth position in the global mobile market. However, as the market shares of BlackBerry OS and Symbian (both currently ranked above the Windows Phone) continue to decrease, the Windows Phone might soon overtake them and gain the third position, though even if it managed to do so it would still be far below Android and iOS (appendix 2).

Similarly to the iOS devices, the Windows Phone supports only a few screen resolutions (table 4).

	Screen resolution (px)	Screen density (ppi)
Windows Phone OS 7.5	480 x 800	252
Windows Phone 8	480 x 800, 768 x 1280, 720 x 1280	233

Table 4. Screen sizes and densities of Windows Phone devices. Data gathered from [19].

Windows 7.5 uses the Internet Explorer Mobile browser that is based on Internet Explorer 9. [19] Windows Phone 8 has Internet Explorer 10 as its default browser. [20]

2.4 Mobile Context

For a long time mobile context was considered to be very distinct from the normal desktop context. It was typical to assume that mobile activity happens only when the user is on the go and most probably in a hurry. [21, 234] These old mobile use cases were used to justify the limited content on the mobile sites compared to the desktop sites. For example, if a user is looking for restaurant on his mobile phone, it was typical to assume that he would only be interested in the restaurant's location and contact information. The available technology in the beginning of mobile era also stimulated this mentality since mobile devices then had small, low-resolutions screens and the input methods were awkward and the networks slow.

But since then, the technology involved has continued to improve and mobile devices have become capable of delivering the full experience. [21, 234] As mobile engagement is continuously increasing (figure 3), the old assumptions regarding the needs and preferences of mobile users are no longer relevant. For instance, eBay's mobile sales rose from $600 million in 2009 to 2$ billion in 2010 ad 91 bids are made every minute via eBay's mobile application. [22] Some Web companies are focusing almost entirely on mobile users. The mobile photo-sharing site Instagram – where signing up is only possible through their mobile app – got one million users in just three months. [23]

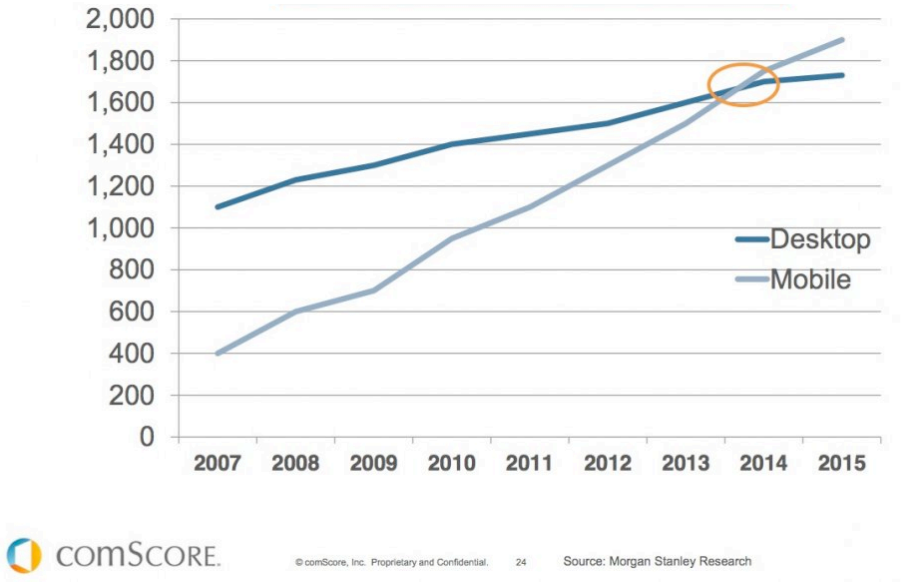


Figure 3. Global mobile vs. desktop Internet user projection. Copied from [24].

These days users will do anything they can on the mobile, when it is the most convenient option for them, and thus the mobile experience is no longer just a second-best option to take to when no primary options (such as a desktop) are available. Compete's Quarterly Smartphone report from 2010 shows how much mobile usage has changed in the recent years and how it has become an integral part of our lives. [25] Of users:

- 84% use smartphone at home
- 80% use smartphone during miscellaneous downtimes
- 69% use smartphones while shopping
- 62% use smartphones while watching TV
- 64% use smartphones at work

The most interesting thing to note from these statistics is that most mobile activity appears to take place at home, which is completely different from the older assumptions of mobile activity mostly taking place on the go. Since mobile devices are used everywhere it has become very hard to optimize the user experience for a certain context as there is limited information on the user's intent. [21, 235] Kadlec suggests considering the "user's behavioural history and location as well as the time, weather, nearby locations, the proximity of other users and the user's movement as factors when trying to discern the user's intent" [21, 236].

Knowing the user's context can be a powerful tool for improving the user's experience and providing relevant information, if there is enough information on the user's needs. For example, a site can use a user's location to optimize its content to primarily show information on what the user can find close by, such as special offers in a store where the user is located.

Context is not dependent only on what kind of mobile device the user has, but on other factors as well. Finck identifies the following key aspects that comprise context:

- User: who is the user and what are his needs?
- Task: what task is the user trying to complete?

- Environment: what is the user's environment?
- Technology: what devices does the user have and what are their capabilities? [25]

This shows that context is a complicated area and many aspects have to be taken into account when determining a user's context. Also, mobile and desktop contexts can no longer be defined separately as they are becoming increasingly interconnected.

2.5 Mobile Users

Mobile users are primarily focused on finding information. However, Nielsen's study on mobile usability showed that users spent more time on performing the same task in 2009 than in 2000, when WAP was in use, e.g. checking the current, local weather forecast. [26, 45] The longer time required for the task now means that in comparison to the earlier, much simpler WAP-sites, some current sites are not yet optimised for mobile usage and that mobile usability for high-end phones still needs to be improved.

Google breaks down the mobile users into three behavioural groups: repetitive now, bored now and urgent now. [27] Similarly, Clark has identified three main mobile behaviours: micro tasking, "I'm local" and "I'm bored". [28] Wroblewski suggests that based on these behaviours mobile usage can be categorised into several task groups:

- Look-up/find (urgent/local): the user needs to find some particular information right away.
- Explore/play (bored/local): the user has some time to kill or just wants to be distracted for a while.
- Check-in/status (repetitive/micro-tasking): the user needs to keep track of changes or updates.
- Edit/create (urgent change/micro-tasking): the user needs to do something immediately. [29]

While it might hard to know why someone would use a mobile device, aligning the structure of a website or an app with these tasks, can help to understand a user's mobile needs. However these behaviours can be equally applied to the desktop user as

well. Therefore information prioritisation is needed for both mobile and desktop and a good rule of thumb is to put focus first on content and then on navigation. [29]

According to a study conducted by the Online Publishers Association, 54% of today's mobile users who own multiple mobile devices prefer a smartphone to a laptop or desktop for at least one of a range of activities. [30] Furthermore, smartphone users are active cross-platform users: 84% of them have watched TV while using a tablet or smartphone and 64% have also been using a desktop computer in addition to a TV and a mobile device. [30]

Mobile users are generally looking for something specific when browsing. Current statistics indicate that mobile users spend 3 to 5 minutes per mobile browsing session. Mobile application design should accommodate this usage pattern by prioritising the easy navigation and fast loading. [17, 256] Nowadays, users get easily frustrated if they can't quickly accomplish a task on their smartphones. If the mobile application is not usable or is slow, people will look for alternative ways to access the information they need.

2.6 Mobile Applications Fragmentation

In the context of mobile applications, fragmentation means the lack of possibility to create one application and run it as intended on all operating systems that are suitable for the application. Device and platform fragmentation is one the biggest issues in the mobile development area (figure 4). [31, 1] Due to the fragmentation, developers are forced to create separate versions of the mobile applications for each mobile platform and a device.

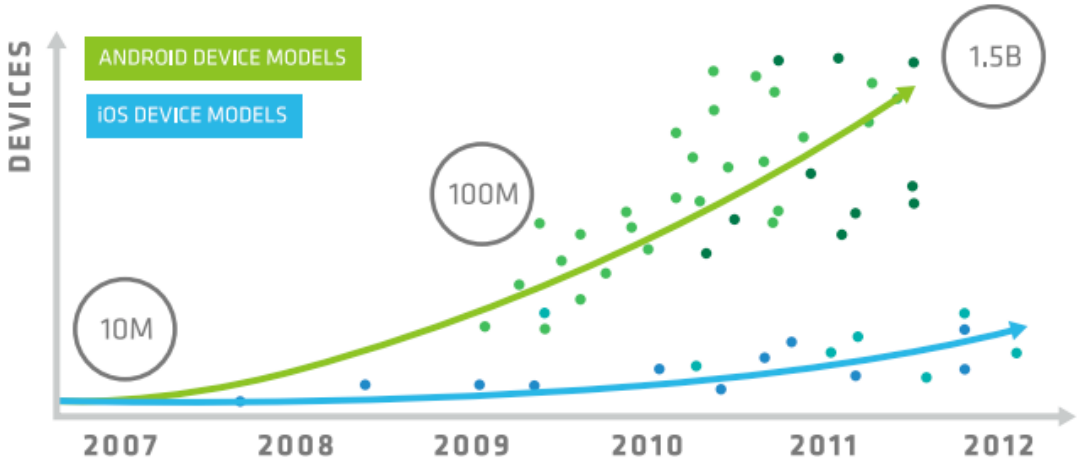


Figure 4. Android and iOS device fragmentation. Copied from [32].

Rajapakse identifies the following key factors for mobile applications [31, 1]:

- Hardware diversity: different screen parameters, memory size, processing power, input methods, connectivity option and additional hardware.
- Software diversity: mobile platforms diversities, implementation diversity (specific quirks and bugs)
- Device features variation
- User-preference diversity: language, accessibility
- Environmental diversity

Fragmentation affects all stages of mobile application development, including project management, business requirements planning, design, implementation, testing and distribution. [31, 2] Therefore the entire process of creating mobile applications becomes much more complex.

This makes it difficult to for developers of mobile applications to choose which platforms to support. Developing for one platform can be difficult enough, but now developers not only have multiple operating systems to consider, but also multiple device types as well. As the number of mobile devices drastically increases every year (figure 3), it is becoming an important necessity to be able to provide the content to all mobile users despite the differences in platforms and hardware.

While developers have to decide which platforms support, they also have to choose what type of mobile application to build: a native application or a mobile web application. [2, 2] The mobile development community has been debating for long time which approach is better.

A native application is an application that has been specifically built to run on a device's operating system (figure 5). [33, 5] The application creator has to develop and maintain a separate application for every target platform and different devices type (e.g. smartphones and tablets). Due to the binary nature of native applications, there are only few tools that can be used for developing native applications. [17, 257]

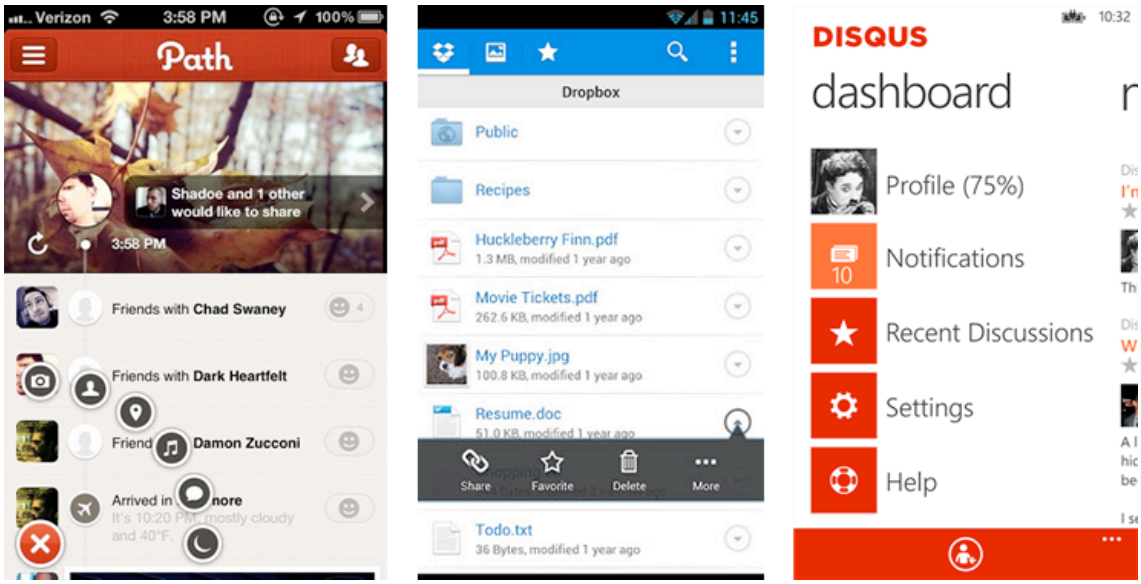


Figure 5. Examples of native applications: Path for iPhone, Dropbox for Android and Disqus for Window Phone. Adapated from [34; 35; 36].

In 2010, the Global Intelligence Alliance conducted a study among 87 developers, publishers and service provider companies on the best approach for delivering content over smartphones. According to this study, companies that followed the native application path comprised the largest share of all surveyed companies – 44%. [33, 12] The main reasons for choosing this approach were:

- Ability to build advanced and usable user interfaces
- Full access to the device's hardware
- Application stores are known and proven distribution channels

- More knowledge in non-web development technologies, such as C++, Java
- Better performance and fit for enterprise environments

Besides native applications, the other way is to create a mobile web application, where all the software is loaded every time it is accessed through a web browser (figure 6). [33, 5] Since the application is accessed through a mobile browser, the same application can be used on almost any mobile platform that supports mobile browsing.

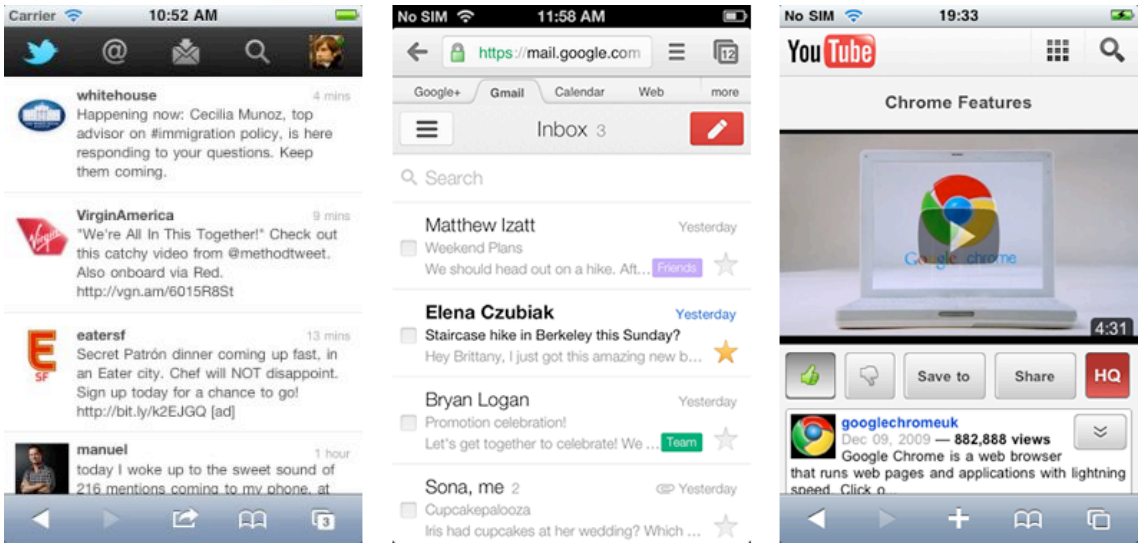


Figure 6. Mobile web apps: Twitter, Gmail, YouTube. Adapated from [37; 38; 39]

There does not seem to be a clear definition of what constitutes a mobile web application. One of the definitions states that a mobile application is a web application run through a browser and utilizing a mobile device's specific functionalities. A mobile web app is also not a mobile web site.

The Global Intelligence Alliance's study showed that 22% of the companies choose to develop mobile web applications mainly for the following reasons.

- The same application can be used on a number of platforms, thus reducing development complexity and costs.
- Applications creators have control over the distribution themselves and do not have to go through the approval process of app stores.
- Usable interfaces can be created utilising web technologies.

- Companies had more in-house knowledge on web technologies rather than on programming languages needed for the native application development.
- Web applications can be released and updated much quicker since there is typically only one code base. [33, 13]

The study also showed that 35% of companies chose to provide both types of applications. Development of both native and web applications did not cause significant increases in costs for those companies and they were able to offer better portability across platforms. [33, 14] One interesting indicator in favour of mobile web applications was that 30% of the respondents with both application types in use had a double increase in usage amounts compared to the companies using only native applications. [33, 16] This proves the lower accessibility threshold of mobile applications, as they can be used immediately without any unnecessary installations. This is especially important for new users, who might want to try the application first before installing it, or for casual users, who use the application very rarely.

Nielsen argues that at the moment "native applications are the best approach compared to mobile web apps as users perform better with applications than with mobile web sites" [26, 34]. In his studies users have had a 64% success rate with mobile websites, while app users have scored at 74%. This success rate refers to the amount of users who were able to successfully complete given tasks during the study.

On the other hand, based on the statistics from the countries with the largest amount of smartphones, mobile apps usage and mobile browsing happen at almost the same rates (figure 7).

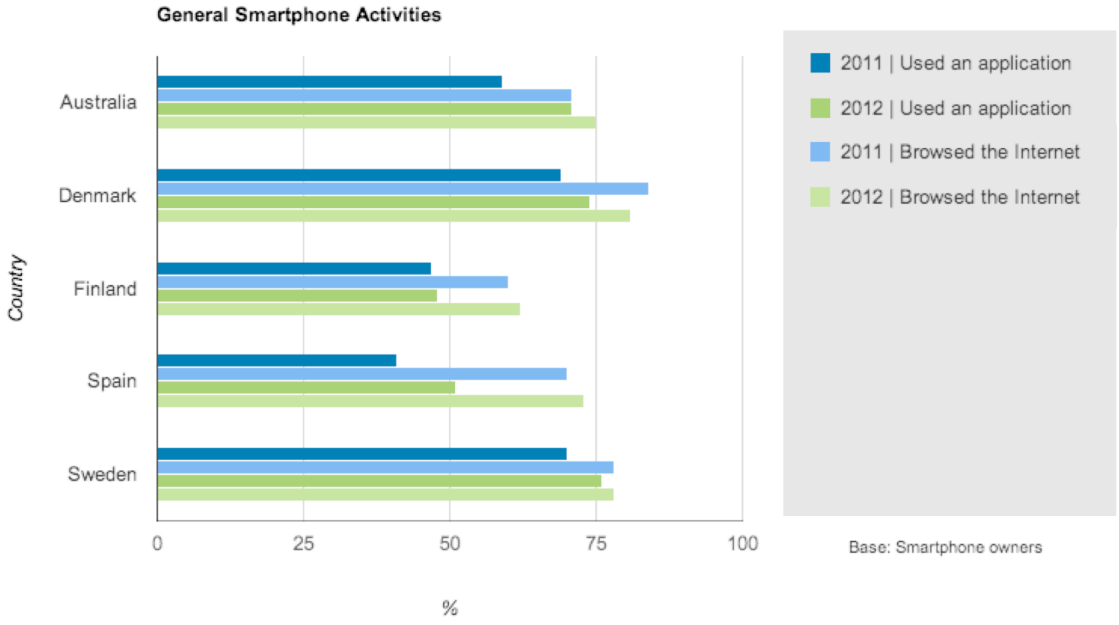


Figure 7. General smartphone activities in 2011 and 2012. Adapted from [40].

While in content categories like news, users prefer to read news on the Web, rather than use a dedicated app from a publisher; native apps usage is nonetheless not declining. [33] For example, 96% of smartphone users downloaded apps in 2012. [30] Perhaps one of the reasons for this is that native apps offer a far better experience than mobile web apps, especially on less powerful devices. [2, 3]

Gemmel makes the argument that different frame of interaction influence app usage. [41] A native app has two frames of interaction: a user interacting through the device with the app. The user sees a device with a status bar showing signal strength, time and battery indicator and then the app itself. With web apps there is a third frame of interaction, namely the browser's interface or "chrome" as it is sometimes referred to. The user's cognitive load increases when there are more unrelated interfaces visible and in this case chrome further increases that load. On smartphones this is particularly noticeable due to the smaller screen estate.

Another issue with mobile web app is separation of concerns. Running an app within a browser contradicts one of the principles of computing devices, i.e. the rule "one tool

per task" [41]. Gemmel argues that people are used to dedicated tools and thus having an app within an app makes it harder for users to focus on their tasks. [41]

The third issue with mobile web apps is that native apps integrate better with the platform they are running on and to the users they feel like a part of it. Users consider apps as especially tailored and targeted entities. [42] For non-technical consumers, the technology behind the apps does not matter. Instead, what matters to these users are the tailored experience and the ability for the user to effortlessly accomplish their tasks.

Each of the mobile application development approaches has its strong and weak points. But the bigger question, as Hales points out, is not which programming methods is best, but what can be achieved until web technologies, such as HTML5, become on the same level with native approaches in terms of functionality and user experience. [43, 19]

3 Developing a Cross-platform User Interface

In this chapter I will present various approaches for developing cross-platform web applications with a focus on a user interface implementation. I will also discuss what needs to be considered when designing an interface that should work on various platforms.

3.1 Cross-platform Development Approaches

Mobile application development has become split into two paths and some service providers and developers choose to follow either only one of them or both. However, the ideal approach would be to have one application that works on all required platforms — a cross-platform application. [44, 1] Having only one application and one code base to maintain would help to reduce the development time and costs. Maintaining existing version and releasing updates and would also be significantly simplified.

Cross-platform development might seem like an easy answer to the fragmentation problem, but it comes with certain constraints and challenges. In contrast to native application development, where the actual development process is well defined by the platform vendors, cross-platform development suffers from too many development approaches and tools. For the new developers it might already become the first issue to address: which development approach to take. Therefore it is important to understand when each approach can be used and what are their drawbacks and strengths.

Raj identifies four development approaches for building a cross platform mobile application. [44, 1] With further classification by Friese these approaches can be categorized into three groups with the following sub-categories [45]:

- Web approach
    - Mobile web applications
    - Client-side web applications

- Hybrid approach
    - Hybrid applications
    - Interpreted applications
- Cross-compiled approach

Web Approach

Mobile Web Applications

A mobile web application is a web application built using HTML, CSS and JavaScript and run through a web browser. Like in a typical web application, data presentation is done on the client side and a server side backend is used to handle the business logic and provide the data access (figure 8). [45] Mobile web application does not require any plugins to be installed since it can be run practically on any device that has a web browser. [44, 1]

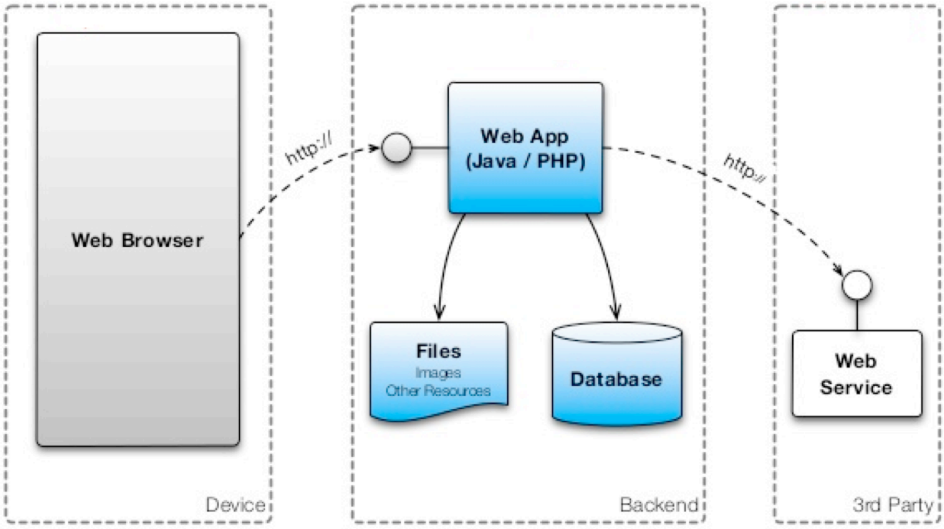


Figure 8. Mobile web application architecture. Adapted from [45].

Mobile web applications do not emulate the interface of the native applications and have the look and feel of a website. However for the layout to work on different mobile devices, like a smartphone or tablet, the content has to be adapted to different screen resolutions for mobile users. [46, 16]

Pros

- Mobile web applications can be immediately accessed by a wide audience, as there is no installation needed. Additionally, the application can be accessed also on a desktop browser, which further extends its accessibility.
- Updates can be easily distributed since there is only one code base hosted on the server and no updates needed on the client side.
- The same user interface can be reused for different platforms.
- Users can easily discover web applications through a search engine and can share them with other people.

Cons

- Unlike native applications, mobile web applications cannot be distributed via mobile app stores. This makes the monetization process more difficult since app stores already have a streamlined sales process and a ready distribution channel.
- Mobile applications rely heavily on network connectivity and thus interruptions in the network may prevent access to the application, which in turn can lead to a poorer user experience.
- Limited access to the device's hardware. With the availability of new device APIs in HTML5, some functionalities, such as geolocation, can already be accessed in many mobile browsers. Some, like camera and microphone access, are only now becoming available in desktop web browsers, but are not yet available in mobile browsers[1].
- It can be challenging to make the web application's user interface adapt to different screen sizes and resolutions and nonetheless maintain a good user experience.
- No full screen mode.
- Even though mobile browsers are improving all the time, developers still have less control over how the application will look and work on the user's device.

1 http://caniuse.com/#feat=stream

- The application's performance closely depends on how fast and standards-compliant the device's browser is.

Client-side Web Applications

Mobile web applications can utilize JavaScript to render the user interface and perform some logic on the client side. JavaScript communicates asynchronously with the backend, containing only central application logic. This eliminates the need to reload the entire application every time a server request is made. A database might be also used on the client side to store the data cache (figure 9). [45]

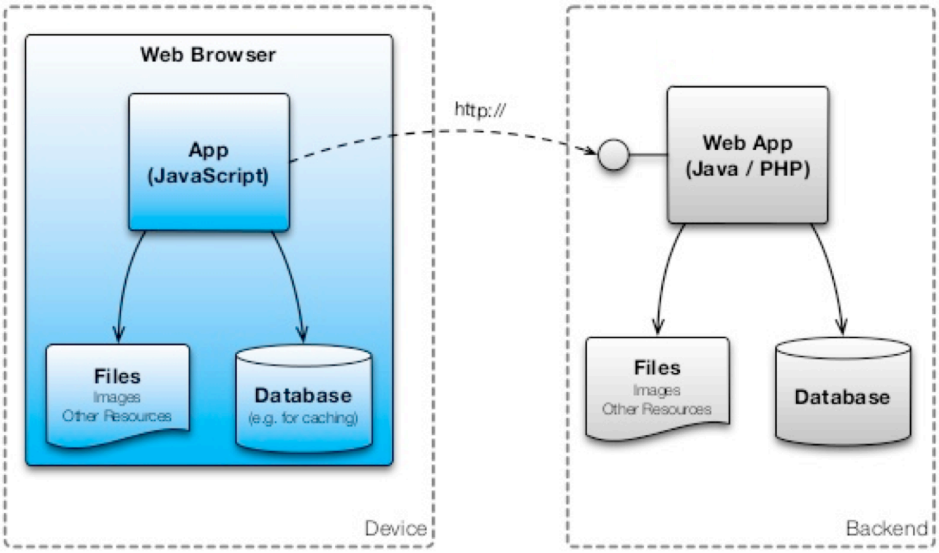


Figure 9. Client-side web application architecture. Adapted from [45].

Client-web applications can utilise UI toolkits like jQuery Mobile to get the look and feel of native applications. UI toolkits provide CSS and JavaScript based interface components similar to the native ones used in iOS and Android. Sometimes even a complete Model-View-Controller framework, for example Sencha Touch, is used on the client side to handle views, client side logic and data manipulation. [43, 52] In the future, this will probably be a practical way to build entire mobile applications.

Pros

- All the same advantages as in mobile web applications.
- The user interface is close to what can be achieved in native applications.
- Better performance due to asynchronous data access and local database caching.

Cons

- UI toolkits typically target only one platform, so UI might look similar to iOS even on an Android device. Therefore, UI adjusting might be required to make the application fit the standards of each target platform.
- Access to hardware features is limited.

The Hybrid Approach

Hybrid Applications

Hybrid applications are cross-platform applications developed using web technologies, and wrapped inside a native container (figure 10). [2, 3] Hybrid applications rely on a browser for rendering the client views and the wrapper exposes the device's hardware, such as camera, microphone, contacts through an abstraction layer as shown in figure 8. Hybrid applications can be used as a standalone app and it can utilise the server backend. However, hybrid applications are one step closer to native applications and therefore must be installed before they are available to the user. [47]

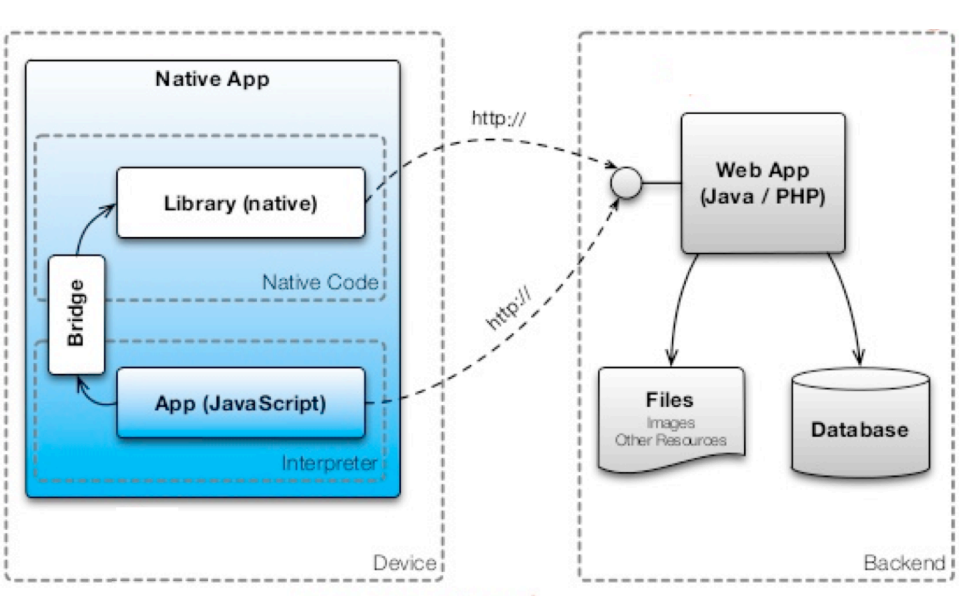


Figure 10. Hybrid application architecture. Adapted from [45].

Pros

- Hybrid applications can be distributed through an app store, which simplifies the monetization.
- Hybrid applications can utilize many of the device's hardware features and thus developed applications can be much more diverse.
- The performance with hybrid applications is better than with web applications, as hybrid applications are powered by the device's computing capabilities.
- User interfaces can be reused on the different platforms by taking an advantage of the native platform features.
- The application can be run full screen. [42, 2]

Cons

- Hybrid application performance is still lower compared to native application, where for example memory usage can be optimised even further. [42, 2]
- A native abstraction layer can also produce cross-space communication vulnerabilities and platform specific execution of JavaScript can cause certain issues.

- It is possible to develop a user interface to look like a native one, but it still might lack the feel and responsiveness of the native application. [45]
- Hybrid applications cannot be accessed from a desktop browser. Though some code can be reused, a separate application has to be made for the desktop browsers. [45]

Interpreted Applications

In the interpreted applications code is deployed to the mobile device and it gets executed during the runtime. [44, 3] An abstraction layer provides access to the hardware's features. The user interface is built with the native UI elements and the application logic is handled in a platform-independent way using a set of commands in XML or another description language (figure 11). [48]

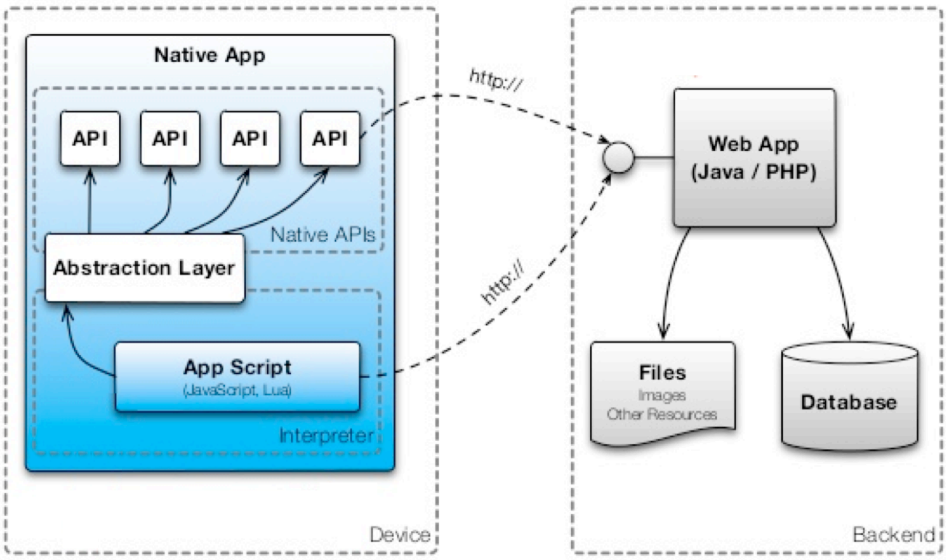


Figure 11. Interpreted application approach. Adapted from [45].

Despite a number of benefits with this approach, a major shortcoming is that development is tightly tied to the features provided by the development framework [48]. This might lead to limitations in terms of providing new features available on the platform, if the framework itself does not support those new features.

The Cross-compiled Approach

In cross-compiled approach code is written in one programming language and a cross-compiler compiles the code into the native code for each target platform (figure 12). [44, 3] With this approach a native application can be produced from a single code base.

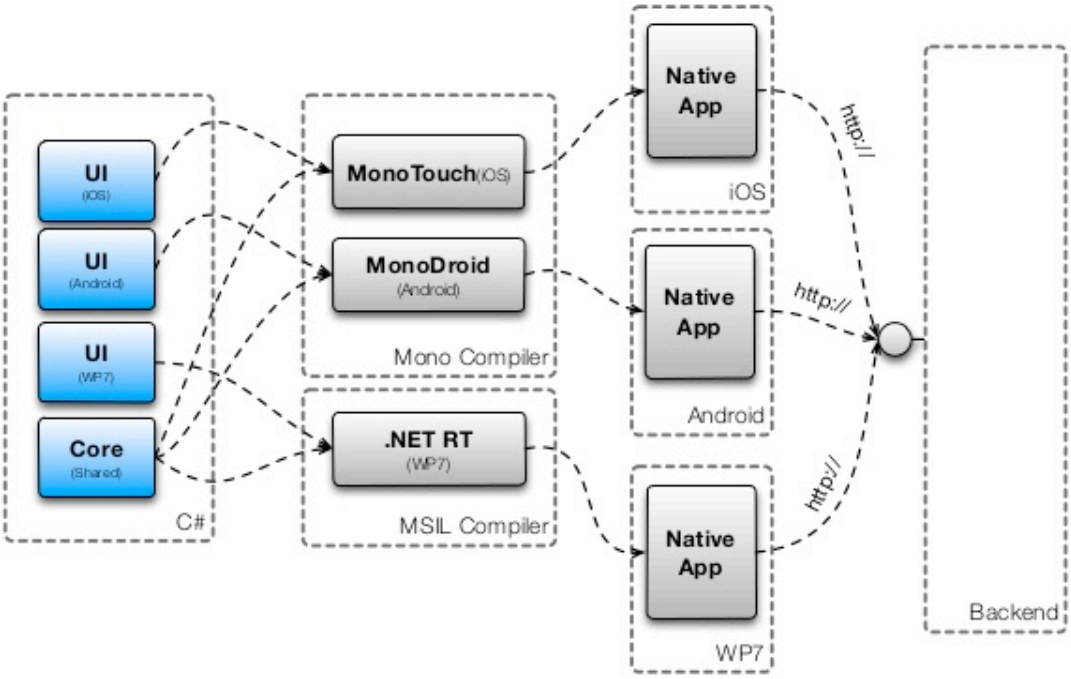


Figure 12. Cross-compiled approach using Mono compiler. Adapted from [44].

The biggest advantage in this approach is the access to all the device's hardware and provided features, i.e. the same as in native applications. Also, the performance is the best compared to all the previous approaches.

Among the main disadvantages is that none of the currently available cross-compiled development solutions are production-ready yet. Also, this approach is not suitable for complicated applications, as compilation process can become very resource consuming. [48] Existing knowledge of web development technologies cannot be applied in this approach, as the code has to be written in C# or a similar programming language. Each of the existing cross-platform approaches comes with own strengths and weaknesses. Choosing a suitable method depends on the specific application requirements. Nowadays, web standards specifications, like HTML5, are rapidly

developing with the active participation of the open source community towards a more open web. New device APIs, that allow accessing a device's features from the browser, have lately been actively developed. Writing one application by using web technologies and running it anywhere is starting to make more sense. [43, 20]

Responsive Web Design

During last years, responsive web design (RWD) has become a popular approach for developing layouts that adapt to the resolution of the user's device and a browser's window size (figure 13). [2, 7] Technologies used in RWD have already existed before, but Ethan Marcotte was the first one to use the term *responsive web design*. He has outlined a set of techniques for implementing RWD and the key ones are:

- A flexible grid for making sure that the underlying page grid scales with screen resolution rather then using fixed pixel size.
- Flexible image for images that will scale with the flexible grid.
- CSS media queries to serve CSS styles tailored to suit a range of screen resolutions and types of devices. [53]

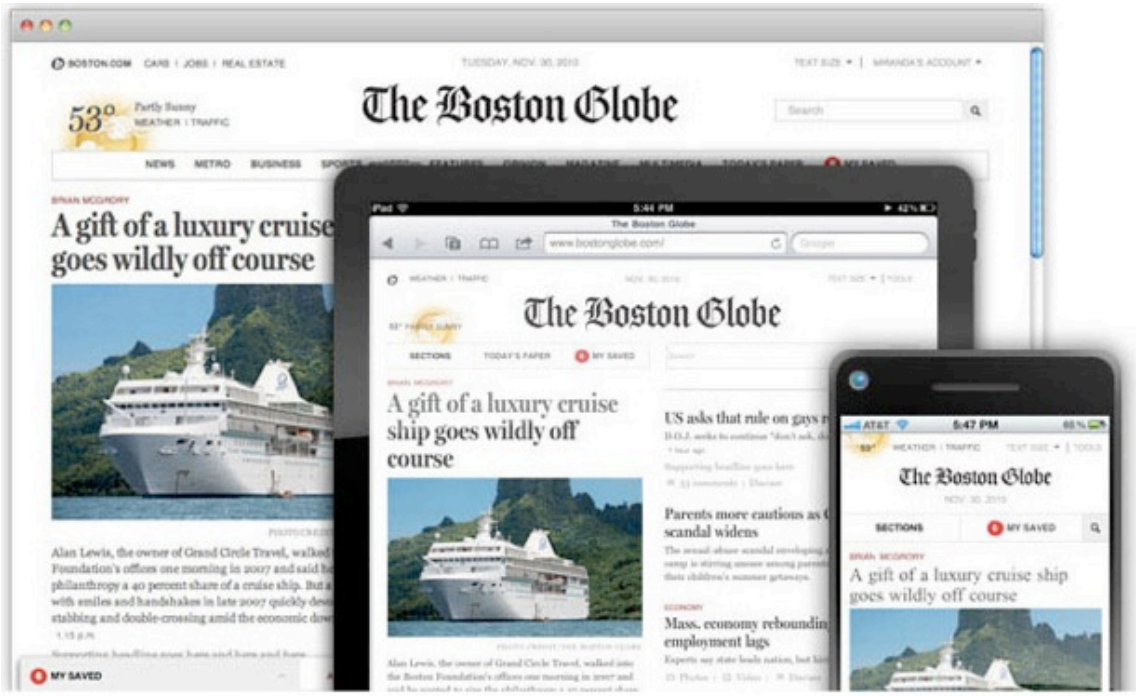


Figure 13. The Boston Globe site on a desktop browser, a tablet and a smartphone. Copied from [54].

RWD uses client-side feature detection to determine available screen capabilities and adapt the layout accordingly. RWD is most useful for designing the layouts, but some have extended it to interactive elements as well, although this often requires JavaScript. Responsive web design allows using a single URL structure for a site, thereby eliminating the need for a separate site.

However, responsive design is not enough for building a mobile friendly site or a mobile web application. Marcotte himself admits that the best approach for building a mobile site depends on the project. "But most importantly, responsive Web design it is not intended to serve as a replacement for mobile Websites" [53, 108].

One of the main issues with RWD is that all the content is always served to the browser and using media queries the layout is then adapted to the device's screen size by hiding and revealing parts of the page. In a mobile application where page loads are a key factor, downloading all the resources, some of which the users won't even see, might not be the best approach. However, with planning for mobile first, most of these issues can be resolved.

Mobile First Responsive Design

Mobile First Responsive Design follows RWD principles from the opposite side to address some of the challenges with designing for mobile devices. Instead of starting with a desktop site, you start with the mobile site and then progressively enhance the experience to devices with larger screens. [46, 20]

Mobile first design offers a number of solutions for mobile design. One of the problems is loading large images for mobile users. The recommended approach for mobile first design is to initially show mobile optimised images first and then replace them with larger versions, if the browser in use is not from a mobile device.

A main benefit of using the mobile first design is that it helps to remove unnecessary clutter from the desktop versions of websites. [46, 20] On the downside, mobile first design has these issues:

- It does not facilitate content adaptation
- A desktop site has to be designed from scratch
- It may force constraints on the interactions for desktop users

Server Side Adaptation

Server side adaptation has been used since the beginning of the mobile web and relies on a device detection library to determine the device accessing the site. [46, 19] Once the device has been identified, the returned content is matched to the device's capabilities. This technique gives most precise control to the developers on the delivered content. Since the content is adapted upon request, server-side adaptation allows providing a contextually appropriate experience for mobile and desktop users. Due to lack of content adaptation on the client side, the delay is limited mostly by network speed.

Responsive Web Design and Server Side Components

According to Podjarny, mobile views in 86% of the responsive websites weigh as much as desktop view. [21, 102] In reality those sites, even though showing the smaller mobile views, still loaded the full website content. This is the main cause for large downloads.

Responsive sites return the whole HTML and for smaller screen sizes some parts are simply hidden using CSS rule "display:none". All site scripts are still loaded and executed. This means that the browser even on mobile devices has to download and evaluate all the content. [21, 102]

Similar problem is also with images. Responsive web design commonly uses fluid images to adapt to different screen sizes. In practice this means that the desktop-sized image is downloaded and then resized by the browser based on the screen's size. [21, 102]

The third issue comes from excessive DOM. Since the whole page HTML structure is loaded, the resulting DOM might become much more complicated than what is actually needed for that particular device. [21, 102]

According to Compuware's survey on global mobile user's expectations, 71% of mobile users expect mobile sites to load as fast or faster than desktop sites. Also, 57% of global mobile Web users had a problem accessing a website in 2012, and 47% had a problem accessing an app on their phone. Also, over 80% of mobile users claimed that they would visit websites more often on their mobile phone if the experience were as fast and reliable as on a desktop. [55]

The average page size is also increasing and is by now over 1 Mb. If the trend continues the same way then the average page size might go up to 2 Mb by 2015. [56] The main causes for huge page sizes are images and third-party scripts used for social sharing, analytics and tracking. Figure 14 provides a comparison of page sizes by content.

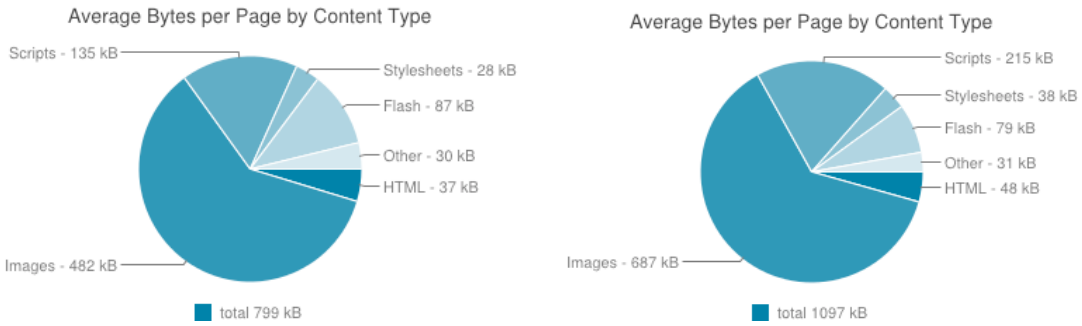


Figure 14. Page size comparison by content type between July 2011 and July 2012. Copied from [57].

Browsing bloated sites can also be expensive for users who do not have a fixed-fee data plan for their mobile subscription and who have to pay for the amount of transferred data. Furthermore, a big amount of research has shown that slow and heavy sites cause users to spend less time on a site, browse fewer pages and purchase less. [56]

Client-Side and Server-Side HTML Generation

JSON and XML provide an efficient way to send data to Web applications, leaving the client side to handle the HTML generation. [43, 64] JSON is commonly used just for data deliveries and therefore it uses much less bandwidth compared to complete HTML documents. After JavaScript and markup templates have been downloaded the first time, it is much easier to manage the resources and rendering of the markup. The client-side approach has the following advantages:

- A better user experience
- Smaller network bandwidths
- Offline capability

However, parsing JSON and generating HTML on the client side requires more memory and processing power compared to displaying server rendered markup. Another major downside is security: WebStorage used for storing data downloaded in the browser does not offer any security. [43, 64] Therefore, in some cases, a server-side approach is a more feasible solution, especially when creating applications that require constant network connectivity. [43, 65] Among the pros of the server-side approach are:

- Better security
- Reduces processing power on the client side
- Improved load capabilities

When developing a mobile web application it is common to produce a solution that has both server-side and client-side processing. Code organisation and distribution in this kind of application can be improved by utilising a client-side Model-View framework, such as Backbone.js, Spine.js or Knockout.js. [43, 65] The main benefit of Model-View frameworks is that they provide a structure to the client-side code and force a clean separation of concerns when fetching from a server-side API.

3.2 Web Standards

HTML5 and Device APIs

HTML5 has become one of the main foundations for creating cross-platform web applications. HTML5 is a fifth revision of the HTML, a markup language used for structuring and presenting content on the web. Even though the first draft of HTML5 specifications has been made in October 2009, browsers have already been implementing HTML5 support before that date. [49, xvi]

Among one the main goals of HTML5 is to evolve the language for easier development of web applications. [49, xii] Therefore, HTML specifies new Document Object Model (DOM) APIs for drag and drop, server-sent events, video, audio, location and many others. These new interfaces are exposed to JavaScript via objects in the DOM and make it easier to write Web applications that follow the specified standards. This is why HTML5 is becoming a preferred solution to create to cross-platform applications for iOS, Android, WebOS and other platforms with good browser support. [46, 15]

One of the breakthroughs in HTML5 is the ability to access a device's hardware from JavaScript using Device APIs. As a result, developers can create HTML5 applications that have similar capabilities as native applications. Device APIs are native code implementations of hardware access in a device's browser. Touch and orientation events give us access to important sensors for input. HTML5 audio and video tags allow playing multimedia without the need for plugins, while taking full advantage of hardware acceleration. Using Web Storage, applications can be developed to have offline support. Web Workers allows for parallel execution, maximizing all of the cores of the CPU. Table 5 shows browser support for client-side APIs that have been finalized by W3C and are considered as a base for building for modern mobile web apps. [2, 15]

Browser	Geolocation	WebSockets	Web Storage	Device Orientation	Web Workers
Mobile Safari	Yes	Yes	Yes	Yes	Yes
Android	Yes	No	Yes	Yes	No
Mobile IE	Yes	Yes	Yes	No	Yes
Opera Mobile	Yes	Mixed	Yes	Mixed	Yes
Mobile Firefox	Yes	Mixed	Yes	Yes	Yes

Table 5. HTML5 support in mobile browsers of the five leading mobile platforms. Copied from [2].

However, not many APIs are supported by all mobile devices, as the W3C Device API group still continues to finalise the standards and device vendors to implement them. For example, Media Capture API, which allows accessing a device's camera and microphone, only recently became available in some desktop browsers, but it is not supported in any mobile browsers yet.

The problem becomes bigger due to inconsistent HTML support among mobile browsers. For example, critical browser features can change from one version of the OS to the next even for one phone model. [50, 6] Also, characteristics may even change for a given device depending on the carrier. As a result, developers cannot fully rely on all HTML5 features and have to check for available features before utilizing them. Feature detection is another big challenge with HTML5 applications and I will talk more about this subject in the chapter 3.14.

CSS3 and Media Queries

CSS3 is the latest version, the third level of CSS specification. It is still under development by W3C, but all major browser vendors have already implemented some of its rules. Due to its modular structure, each module in CSS3 can bring new functionality or features while maintaining backwards compatibility with CSS2. CSS3 introduced a number of significant capabilities for the styling of Web pages: new selectors, better visual effects, animations, multi-column layouts and much more. These new features allow building content-rich web pages with relatively lightweight code requirements, and, most importantly: CSS3 works cross-platform.

Some of these features are not yet fully supported by mobile web browsers and therefore should be used only for non-critical features, though this situation is improving fairly fast – CSS3 selectors are already supported in the latest versions of iOS Safari, Opera Mini and Android Browser.[2]

I have mentioned media queries in the previous chapters when talking about responsive design. Media queries are able to request the physical characteristics of user's device and different stylesheets can be delivered to the devices with matching characteristics. [2, 8] This way the layout of the web page can be altered based the device's properties. Before CSS3 appeared, CSS2 allowed developers to specify only the media type, for example, 'screen' or 'print'. CSS3 has been extended with an addition of media queries support. In June 2012, media queries have become an official W3C recommendation making them a part of the Web Standard. [51]

A media query consists of a media type and an expression that checks for conditions of particular media features. Among common media types 'screen', 'print', 'handheld' and 'projection'. Media features include properties such as 'width', 'height', 'device-width', 'device-height', and 'orientation'. Utilizing media type and features, developers can control when certain styles are applied to the page (listing 1).

```
@media screen and (min-width: 400px) and (max-width: 700px)
{ … }
```

Listing 1: This media query tells that the style sheet is usable on devices with viewport (the part of the screen/paper where the document is rendered) widths between 400 and 700 pixels.

While media queries might be a suitable solution for adapting the layout on the desktop web, for mobile web, where bandwidth is an issue, they might not be the best solution. The problematic situation is caused by several factors:

- All devices are loading the same CSS, JavaScript, images and video, something which causes unnecessary, longer loading times.

2 According to the latest statistics from http://caniuse.com/#feat=css-sel3 checked on 5.8.2012

- All devices use the same initial DOM and that might result in overly complicated CSS.
- It becomes difficult to define custom interactions for each device. [52]

Media queries can be used as a part of a solution for creating cross-platform apps but as UIs for web applications are becoming more complex, we will need to have more flexible ways to customize the UIs for each device type.

3.3 JavaScript and UI Web Frameworks

JavaScript Frameworks

A few years ago, writing complex frontend applications in JavaScript was difficult and often resulted in hard-to-maintain code. [43, 83] Nowadays, there are many frameworks that aid web development using HTML, CSS and JavaScript. These frameworks offer the following key advantages:

- They bring structure and clear organization to the frontend code. [43, 83]
- They help to reduce the network load by minimising the number of HTTP requests for displaying the views. [57]
- They save development time by providing patterns and solutions to common problems. [58]

Some of the frameworks follow the Model-View-Controller (MVC) programming pattern where the application concerns are separated into three parts (Figure 15). [58]

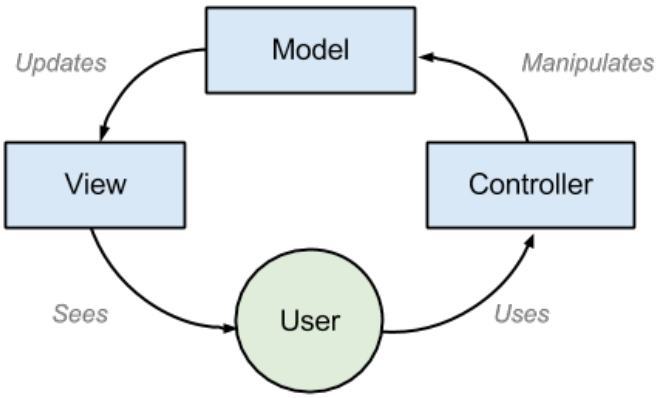


Figure 15. MVC process used in JavaScript MVC frameworks. Adapted from [43, 83].

- Models represent the data and domain specific information and notify other components of the current states.
- Views are used to build the user interface. Views observe the data in the models and display it using the HTML templates.
- Controllers handle the data input and update the state of the Models. Since Views are observing the Models, Controllers do not need to explicitly call the Views to display the changes. [58]

Some frameworks include only a Model-View (MV) part and add their own components in place of the Controller or combine the Controller with the Views. These frameworks are referred to as MV* frameworks. [58] Additionally, there are frameworks that follow Model-View-Presenter and Model-View ViewModel patterns. It is not easy to choose a suitable framework for the project also because there are over 40 different frameworks available nowadays. [43, 83] Among the most well known are Backbone.js, Ember.js, Angular.js and Knockout.js.

When choosing a framework it is important to consider how it handles the data interactions, meaning that UI binding should be declarative and views should self-update when the data changes. [43, 84] Osmani outlines several other criteria to consider when selecting the framework [58]:

- Is the framework mature enough and has it been proved in the production environments?
- Does the framework force the development in certain way or does it let the developer choose how to do things?
- Does it have an extensive documentation and an active community around it?
- What is the overall file size of the framework including the dependent libraries after compression?

MV* frameworks can be useful in cases where the views rendering and data manipulation will be happening on the client side and where backend communication will occur via the data API. For situations where there is minimal amount of client-side interactivity and the backend is responsible for rendering the views and manipulating the data, MV* framework might not be the most suitable solution. [58]

Even though there are many different frameworks for structuring the JavaScript applications Osmani states that one of the most important steps in the selection process is to thoroughly evaluate the available options and try some of them before beginning the actual development.

User Interface Frameworks

There are JavaScript frameworks that focus on providing cross-platform UI for mobile devices and help to deal with cross-device and cross-browser differences. [2, 100] Among the most widely used one are jQuery Mobile, jqTouch, Sencha Touch and Kendo UI, however each of them has its own advantages and drawbacks. [59]

jQuery Mobile, one of the popular HTML5 UI frameworks, is based on a jQuery plugin. [43, 48] The goal of this framework is to provide a user interface system that works on all platforms, including smartphones, tablets and desktop platforms. Currently, it

supports a wide majority of mobile and desktop platforms. jQuery Mobile is optimised for touch input, fully customisable and themeable, provides AJAX-based navigation systems and follows progressive enhancement and responsive web design principles. [60]

jQTouch is a lightweight plugin for Zepto/jQuery libraries. It provides UI components, animations and themes for mobile devices. [61] It is not as extensive as jQuery Mobile and its development is not as active either. One of the main limitations of this plugin is that it works on mobile WebKit browsers, which limits its usage for other platforms, such as the Windows Phone.

Sencha Touch is another HTML5 framework that follows the MVC pattern. It has support for iOS, Android, BlackBerry and other platforms. Sencha Touch provides cross-platform native-like UI components, animation, theming, data stores, touch support and adaptive layout system. [62] When considering a framework, one important aspect to take into account is that Sencha Touch has a specific user interface development model. [43, 52] jQuery Mobile or jQTouch rely on specifically structured HTML markup for turning a page into AJAX-based animated views. However, Sencha Touch is 100% driven by JavaScript, even for creating UIs. While the framework is free, it has a steep learning curve and the documentation seems to be disorganized and not very up-to-date. Also, it is suitable only for mobile applications and not directly suitable for the desktop web.

Kendo UI is an HTML5 framework based on jQuery that is targeted for creating mobile web applications. Kendo UI is different from jQuery Mobile and jQTouch as it is an MVVM framework. Besides UI components, animations and theming, it provides client-side DataSource, a data binding mechanism, touch support and templating. [64] The framework can be used for mobile and desktop web development. It comes with an HTML5 powered mobile UI that automatically adapts to different mobile platforms such as iOS, Android and BlackBerry. The main issue of this framework lies in its licensing – the complete framework with mobile support requires a licensing fee.

One of the issues with most of these frameworks and plugins is that they all come with predefined user interface components that try to emulate the look and feel of the

native apps. While they can be customized to certain extent, in my opinion they create a false impression by trying to look for example like iOS and Android UI components, but in reality being neither. [2, 10] Relying on heavy usage of CSS decorations to emulate the look of native components can slow the browser's ability to crawl a page's DOM tree and thus reduce the performance of the application. [43, 48]

Considering when to use these frameworks depends on the overall mobile web application approach chosen by the developer. If an application is going to be built using a JavaScript framework like Backbone.js or Knockout.js, which already provide MV structure, it make more sense to include jQuery Mobile for handling the UI work, instead of Sencha Touch, since it includes a complete MVC stack.

When choosing a UI framework, Hales also advices that it should [43, 47]:

- Be optimised for touch input and create animations using CSS3 transitions for improved performance.
- Have a consistent browser support on all major platforms.
- Follow the latest HTML5 and CSS3 standards.
- Have an active development community.

3.4 Mobile Design Considerations

Accounting for Device Orientation

Designing for different screen sizes with consideration of the device orientation can be pretty challenging and requires careful planning. On the other hand, changing orientation can be considered also as a benefit, because just by turning a device we get more space for additional layout. Device orientation can be used as a secondary display to show additional information or extend the current layout. [65]

For example, YouTube's mobile application in the portrait mode shows the video and related information like author, comments and rating. When the user rotates the device to the landscape mode, the view changes to the full-screen video player with

playback controls, thus improving the viewing experience (Figure 16). When the video finishes playing, the view changes back to the portrait mode.

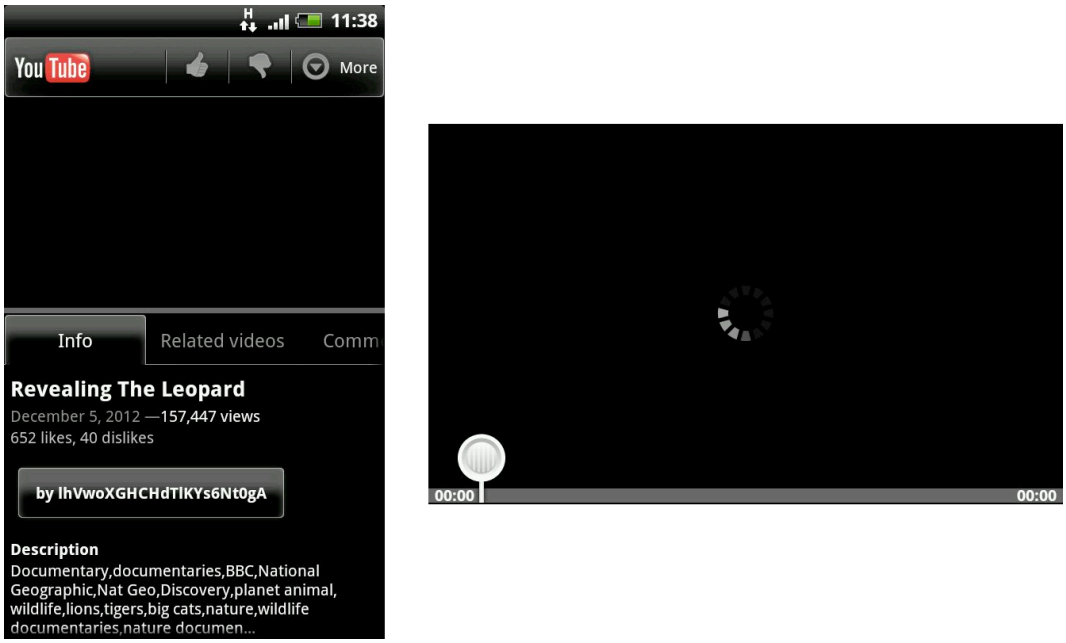


Figure 16. YouTube app in portrait and landscape orientations. Author's own visualisation.

According to Itzkovitch, four layout strategies can be outlined when designing for device orientation [52]:

Fluid – the interface stretches to fit the new screen size, but the interface remains the same.

Extended – the interface adjusts to the screen size by adding or removing interface components based on the orientation mode. For example, in the landscape mode the interface can display additional information.

Complementary – the interface changes to the secondary screen, which shows the related supplementary information. For example, in the portrait mode the application can show a list of data and when turned to landscape mode it shows the data as a graph.

Continuous – rotating the device shows the secondary interface, something, which extends the functionality of the screen in the portrait mode. For example, an

application used as a TV remote control would show a full program guide in the landscape mode, while retaining the possibility to control the playback and change the channel.

Apart from how the user interface would work when the orientation is changed, another important aspect to consider is the default orientation. For most smartphones and the iPad the default orientation is a portrait, while for Android, Window8 and BlackBerry tablets the default is landscape. The latter also applies for desktops. Therefore, the primary orientation of the application should show the default interface and functionality. [52] Also, some devices allow locking the screen orientation, something that might prevent access to some part of the application.

Designing the layout in order for the application to work well with device orientation requires extra work. Nevertheless, orientation can be used to improve the user experience of the application. For instance, an application for watching videos could utilise the landscape mode to show the content fullscreen and prevent, for example, the screen from auto-dimming. Other common behaviour would include displaying a larger virtual keyboard in the landscape mode when writing an email or SMS message.

Accounting for Touch Interactions

Touch devices have unique features in terms of design and usability. Touch accommodates for much more intuitive interactions than a keyboard or a mouse, but to be able to provide them to the users a different design thinking is needed. Fingers are not as precise as a mouse and have bigger touch target areas. Hit zones on user interface elements should be adapted to take that factor into account. Apple's Human Interface Guidelines for iOS recommends making the touch targets at least 44 x 44 pixels big. [66]

Touch-based interfaces can easily become unusable if there is too much content crammed into the view, thus making the selection of individual interactive elements hard. [66] One of the main issues with touch-based input is that touching the element covers it from the user's view. If touch targets are too small or if there is too much content, it makes it difficult for the user to tell which element he has selected.

Some behaviours, common for mouse-based interactions, are not applicable for touch-based interactions, e.g. hovering. [2, 105] Hovering is used widely in the desktop web in order to, for example, show drop-down menus, image captions and link states. Thus, while hovering might work on some platforms, it is recommended not to rely on hover states, as they might not work as expected. [2, 15]

Another important aspect is the placement of the navigation elements and controls. For most comfortable operation the placement of navigation elements should be done in accordance with how people hold and use touch-enabled devices [68]. Smartphone users often use the device with just one hand and type with one or both hands (Figure 17). Also for right-handed people the bias is towards the right side of the device.

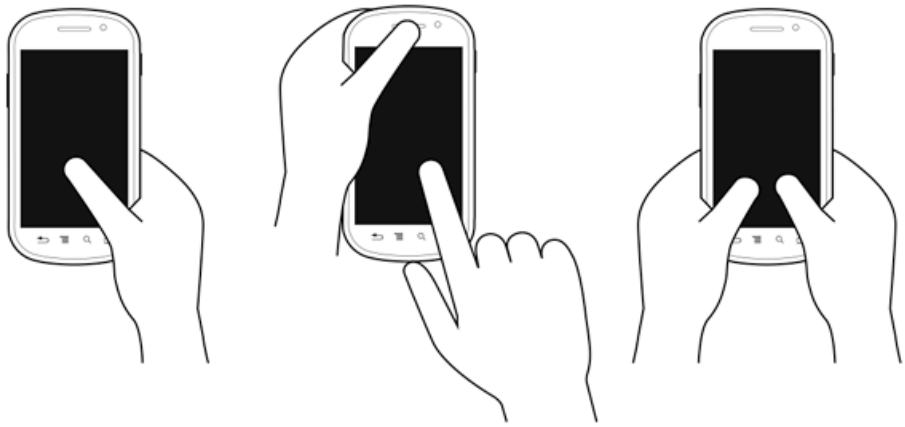


Figure 17. Typical hold positions for the smartphone. Copied from [68].

Wroblewski identifies areas at the bottom of the screen and right hand side as most easy to reach with, while upper area being most difficult as shown in Figure 18. [68]

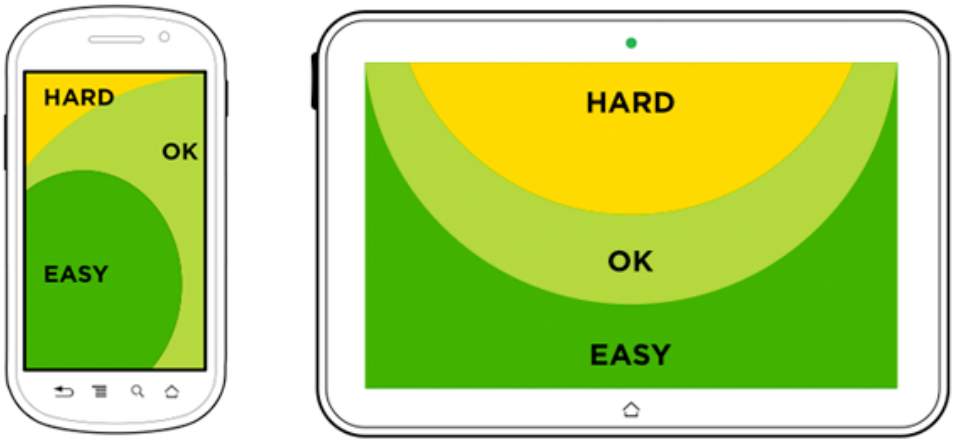


Figure 18. Interactions zones by difficulty on a smartphone and a tablet. Copied from [68].

Tablets are typically held along the sides and typing is done while keeping the device on a lap or a table especially in landscape orientation (Figure 19).

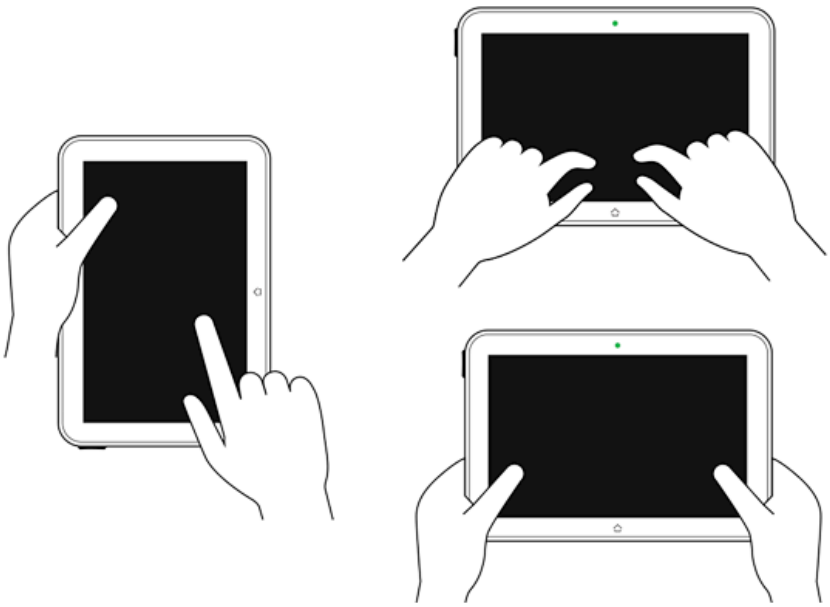


Figure 19. Typical hold positions for a tablet. Copied from [68].

In portrait mode easy touch areas are at the bottom of the screen and in portrait mode at the bottom and sides of the screen as shown in Figure 20.

Figure 20. A proposed placement of navigation for better access on touch-enabled devices. Copied from [68].

Based on these observations, the most ideal place for placing important navigation controls for smartphones and tablets tends to be at the bottom of the screen. [86] However, on most desktop websites, designed for a mouse and keyboard, interaction navigation is located at the top, either on the left or on the right. Therefore, the placement of navigation controls has to be adapted for different devices. Due to limited screen height on mobile devices, positioning fixed navigation controls at the bottom of the screen might not always be a good idea. Instead, page footer navigation can be used to provide quick access to navigation controls with an anchor link at the top.

Even though this might sounds like a more consistent approach for navigation placement, in my opinion, splitting navigation and placing it at the bottom of a bigger screen might be confusing for users. Many studies show that our eyes tend to move in F-shaped pattern when scanning a page. [69] Having the navigation at the bottom of a large screen would then require a long movement in order for the eye to find the menu. Also, people are more accustomed to having navigation on the left or topside of the page.

Among the main considerations for designing touch-based interfaces are [4, 70]:

- Clickable elements should have a sufficient amount of space between them, 20 px or more. Frequently used links and button should have big clickable areas, at least 44 px in width and height.
- Finger gestures should be used for compatible devices.
- Feedback should be shown when the touch selection is accepted.

- Form labels should be added above the input field to remain visible during typing.
- Infinite lists are recommended over pagination. An infinite list dynamically loads new items when the user scrolls a list.
- Text input should have an auto-clear option.
- Bottom-fixed tab navigation is preferred to top-fixed ones since the bottom of the screen is nearer to the user's finger fingers when he/she is browsing.
- Drop-down menus should have clear visual cues.

Graceful Degradation and Progressive Enhancement

Graceful degradation and progressive enhancement are two techniques for adapting the content for mobile devices. Graceful degradation appeared from the need of the design to work on as many platforms as possible using the newest technologies, without excluding the users that did not have the support. [53, 129] The main idea is to serve the best user experience possible and then adapt the site to the capabilities of the device to remain functional. In practice, this means that a website will gradually remove content and features as the viewport's size and web browser features support decreases.

Progressive enhancement is another popular content adaptation technique first introduced about 10 years ago by Steven Champeon and Nick Finck. [46, 18] The idea behind progressive enhancement is to serve a single base HTML page to every device and use JavaScript to build up the functionality based on the capabilities of the device. If the device is from the low-end devices range, JavaScript won't run and the user will get a basic experience. But, if the device is a desktop browser or a smartphone, functionality will be progressively added to the page.

The main benefit of this technique is that it can be used for a wide range of devices and provides an adaptive user experience. Progressive enhancement keeps the logic of determining how to adapt the content on the client side. One of the caveats of this approach is the delay in the progressive build-up that may occur due to a slow network or a slow device. [46, 18]

Device and Features Detection

In order to be able use HTML5 features and device APIs in a web application we must first detect what features the browser actually supports. Earlier feature detection relied on detecting the browser's user agent (UA) string. Once the user agent was known, device's features could be retrieved from device libraries. [4, 330] There are several issues with this method and it is recommended to avoid it if possible. [21, 201] First of all it is not very reliable. Browser vendors sometimes do not include a correct UA string or it may be incomplete. For instance, the userAgent string for the mobile Firefox browser on Android incorrectly provides the information that it is a tablet instead of a phone [43, 66; 70]. Another issue is that information on all devices' capabilities and their UA has to be regularly updated. Device libraries can solve this issue, but the most extensive and up-to-date ones are not free.

The Mozilla organization also states that delivering different HTML based on a browser is a bad practice, since the Web should be accessible to everyone independently of the browser and device used. Browser detection can be useful only for some edge-cases. A better approach is to instead check what device the user is using and then check for the support of the specific feature that is needed. [21, 204] Nowadays, these features can be easily detected with a JavaScript or using frameworks, such as Modernizr, which works by first running quick tests to detect browser features, then creating a JavaScript object with the results and appending feature-named classes to the html element on the page (listing 2).

```
<html class="js canvas canvastext geolocation rgba hsla no-multiplebgs
borderimage touch”>
```

Listing 2. Modernizr applied feature-named classes.

Individual feature detection can be also in done in JavaScript (listing 3):

```
if (Modernizr.geolocation) {
  //Initialize geolocation function
}
```

Listing 3. Individual feature detection in Modernizr.

Feature detection has it own downsides as well. Running many tests can take some time and increase the page load time. Fortunately, Modernizr allows selecting which tests to run and this is a better approach for the production environment. [43, 67] Mozilla also advises using progressive enhancement or graceful degradation. [70]

3.5 Cross-platform Web UI Development

Rapid development of Web and related technologies as well as devices capable of Web browsing has forced developers to also change user interface design and implementation workflows towards much more dynamic and flexible approaches. [71]

Figure 21 shows a workflow process for designing a static interface web application.

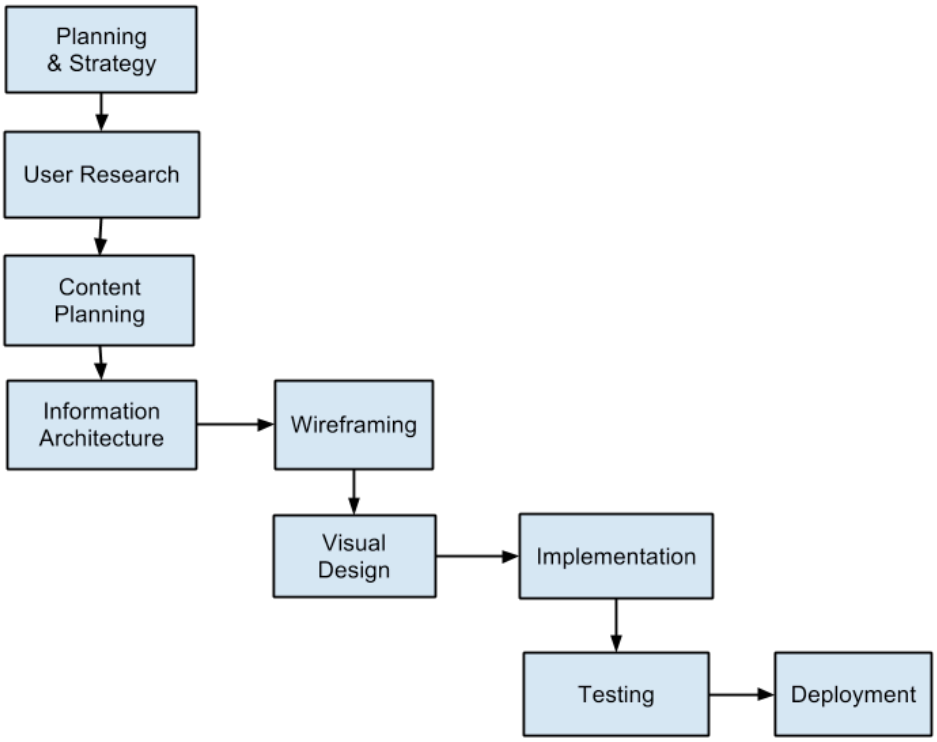


Figure 21. Workflow process for a static web UI. Adapted from [21, 154].

The planning involves research and analysis of user goals and target groups and outlining of the requirements. The content is planned before the design process begins or it might also happen the changes are made during the implementation stage. The design is done by making static wireframes, mockups and visual compositions. The

design of static UI is done around layouts that are aimed for desktop usage. Layouts are made either with fixed or liquid grids, but no considerations are made for other than typical desktop screen sizes. After the design is complete, layouts are passed to developers for implementation. Once the implementation is complete, UI is tested and necessary fixes are made and then deployed to production.

One of the issues with a static UI workflow is that static UI mockups, produced in the design phase, do not show the developer how to handle visual styles with the user interactions, thus leaving much design decisions to the developers. [21, 170]

So far, the design process for the static web has been similar to print design and has even used the same tools. Typically, the entire process followed a linear approach, where the next stage started after the previous one had ended. [21, 172] Due to this single-direction approach any issues that appeared during any of the stages are accumulated and most likely addressed at the end. Another limitation is that no considerations were taken into account regarding platforms other than the desktop.

When developing a scalable user interface for a cross-platform web application the linear approach is no longer applicable, as the issues with different platforms/devices have to be addressed in the beginning of the work. A multi-device environment requires new approaches and a much more flexible workflow. [21, 171]

Designing a scalable UI requires changes in planning, design and user experience processes. Figure 22 shows the process flow involved in developing a scalable UI.

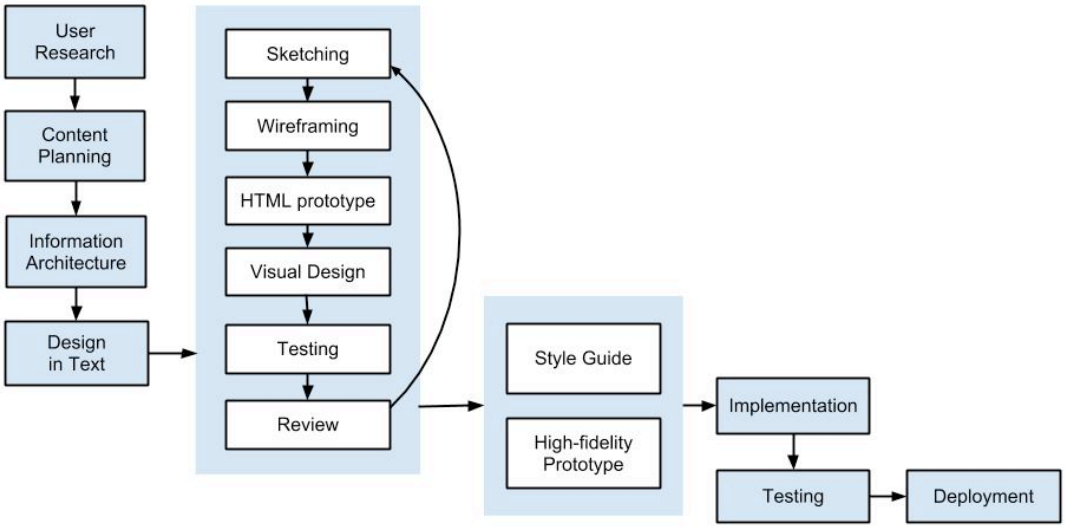


Figure 22. Workflow process for a scalable UI. Adapted from [21, 156].

The process for implementing a scalable UI starts by performing a user research similarly to static UI process. However, more analysis is done for understanding user context due to the multi-device usage. During the planning stage focus is put on planning the content strategy and information architecture. Using a module-based approach, the main content types are identified and prioritized based on the usage contexts.

The major difference between the static and scalable UI processes is that in the latter one the workflow is much more indirect and design and development phases can even be combined. [21, 156] Instead of focusing on creating wireframes and static UI mockups, the goal is to produce a scalable prototype that can be used as a base for the implementation. Essentially the whole design process is an iterative process where each step is repeated until a desired solution is found.

The design phase is started with a design in text format, where the content is identified. In the planning stage then the content is put into simple HTML wireframes without any styles in order to see how the content will behave on small and large screens. This step does not aim to produce a finalized content structure, but it will be used as a basis for further design work.

Sketching is done frequently to test various UI ideas and it is especially useful before developing the prototype further. Prototyping in HTML and CSS is started early, right after the sketching, in order to get a better understanding of how the UI will work on different devices and in order to get user feedback. The goal is to find and address most of all the possible UI scalability issues before building the actual application. Creating an HTML prototype also helps to put the focus on the content and structure and to understand how users will interact with the user interface. [21, 170]

A large part of the design process relies on the continuous testing and updating of the prototype. Initially a low-fidelity HTML prototype is created to show how the layout and interface interactions will work. During the course of the design stage it is developed further into a high-fidelity prototype with a working user interface.

Visual design is also a gradual process and it takes place before and after prototyping. Commonly, the design is started in a graphical editor by creating static detailed layout mockups, but some designers also lean towards doing most of the design in a browser. However, it is hard to see in static mockups how some aspects of the layout will work on different sized screens, for example typography and whitespace. Therefore, it is best to continue designing in a browser as soon as possible. An alternative to building initial static mockups is to create style tiles, which are a design deliverable that is used to communicate the visual direction of an app. Style tiles consist of fonts, colours and interface elements.

Testing is done continuously during the design stage along with the development of the prototype and visual design. It is mandatory to test not only in a browser but also on the actual devices mainly due to a large number of differences between the devices. [4, 435] Testing should also involve some of the users to find out how the functionality meets the users' needs.

During the design phase the low-fidelity HTML prototype is iterated to a final high-fidelity prototype that meets the requirements of the application. Another deliverable, which is made during the design phase, is a style guide. A style guide specifies UI components, CSS styles and HTML markup as well as how they are used and might

also include instructions on how the UI behaves during resizing. Developers use a style guide during the implementation as a reference for building the app's UI. [21, 173]

Once the design is complete and there is a clear plan for how the UI will scale on different devices, the final implementation is started based on the prototype and style guide.

Mobile Usability Considerations

The success of a mobile application depends on the design and performance. Web application should follow the web usability principles and in addition have a consistent look and feel on the target mobile platforms. [43, 19] Each of the areas of the mobile user experience, like content, context and users requires different considerations. [26, 23]

To give users the best possible experience on different screen sizes – in particular tablets – you need to optimize the layout and other UI components for each target screen configuration. With limited screen size, varying bandwidth and short user focus lightweight, CSS-based layouts, content-out design and accessibility get even more important. In addition, all usability principles for the desktop Web can lead to better-designed mobile web application. [26, 7]

Nielsen outlines four main usability issues that affect mobile usage [26, 50]:

Small screens: smartphone screens are relatively small compared to tablets and desktop computer. Users can see a limited amount of information and controls, which makes interaction more difficult. For example, displaying product comparison information can be done only for a few products at the same time and may require lots of scrolling to see the rest.

Input is more awkward: finger-based input is not precise and is prone to errors. Text typing is especially slow and mistakes occur often.

Download delays: loading can suddenly decrease to dial-up speeds or get interrupted when the signal is weak.

Not optimised design: sites that follow desktop usability rules are often not usable on the mobile.

Nielsen suggests the following base practices for designing mobile sites. [26, 20]

- All features that are not necessary for mobile usage should be removed.
- Present only the core content and leave the details to secondary pages.
- Make interface elements large enough to be suitable for touch interactions.

Mobile sites should be adapted for one-hand-usage. Wroblewski advices minimizing the need for the keyboard and resorting instead to tapping, for example, instead of entering a date, let the user choose it from a date-picker. With reduced keyboard usage, visual communication becomes more prominent. Interactions can be more focused by simplifying the process flow and the number of steps required to perform a task. Reveal necessary actions only when needed, i.e. show the main navigation when the user stops scrolling the page or show product details only when the user asks for it. [29]

Also, we need to consider how user flow will work on multiple devices. Users tend to start a task on one device and then continue it later on another one. [71] For example, searching for information on a smartphone while being on the bus and then continuing the search later at home on a latptop. In this case it would be much more convenient if the site, where the search was made, would remember the last user searches. Another example could be taking and uploading a photo with a mobile and then later adding description and tags on the desktop site.

Summary

The mobile web has developed a lot since the WAP days. So have mobile devices. It is becoming the norm to provide parity content for both desktop and mobile users. Whether it is a native or a mobile web application, users expect to access it in the way that is most comfortable for them. Choosing a suitable approach can be difficult and creating a mobile web app that can run on all platforms and all browsers is a big challenge. But, as Hales states, we must aim to build applications that are working across multiple platforms and devices. [2, 9]

While the cross-platform web approach seems to be the most suitable solution when the application should be accessible on many different platforms, it has its own limitations and design constraints. The nature of a mobile application plays an important role in decision-making. For server-driven applications where a mobile client is used to retrieve and send data, a web approach would the most appropriate. Tools and Techniques

Designing for the mobile web requires finding the right balance between the information amount and interface elements, especially when creating a cross-platform user interface. Understanding the user context and content as well as mobile device capabilities and constraints is essential for creating a usable interface and providing a great user experience.

4 Booklio, Design and Implementation

The following chapter describes the design and implementation of the cross-platform user interface of my books management application. The first part of this chapter focuses on outlining the requirements and goals of the application. The second part describes the actual design process and implementation. Since the main focus of the application is on design and implementation of the user interface, backend architecture is discussed only briefly as it is not within the scope of this project. During the implementation I have used some of the techniques described in Chapter 2.

4.1 Overview and Expectations

In order to understand and find out the issues with developing a cross-platform user interface, I have created a prototype mobile web application. The main focus was to develop a user interface that would be usable on smartphones, tablets and desktop computers by adapting to the specifics of the accessed device and retaining the best possible user experience.

Since this project work is not carried out for a third-party, I decided to create an example application that could be used to answer the questions set in this research. Being an active reader I chose to create a book cataloguing application. Another motivation was that most of the leading books cataloguing web services have very basic mobile sites and I was interested in the elements required to build a better service.

The example application, Booklio, is an application that allows users to track the books they own, have read and wish to read as well as view personal reading metrics, check and create book reviews as well as get book recommendations. The application is mainly targeted at avid readers. Due to the specific scope of the work, the final application does not have complete functionality and the application’s architecture has been simplified.

4.2 User Requirements Analysis

Target Users

Before deciding on the features of an application, we first need to understand who the target users are, what is known about them and what they might want from the application to be designed.

One of the popular book discovery web sites, Goodreads, has made a survey comprising input from 1500 members on the topic ”What’s Going on with Readers Today”. [72] This survey showed that 83% of its members are female and the rest are male (17%). The largest age groups are 25 – 34 year-olds (28%), 35 – 44 year-olds (24%) and over 55 year-olds (17%). [72] From these it can be concluded that female adults is the main demographic group. Additionally target users are people who:

- are avid book readers
- have too many books and need a way to manage their library
- enjoy discovering new books
- are interested in tracking their reading or libraries

Application Features

Once the target audience has been outlined, we need to narrow down the feature-set and decide which features should be included and which should left out. This can be a challenging process, therefore, it is recommended to think of the minimum viable product: which features would be essential for the application. [2, 23] The Goodreads survey provides some useful insights into user reading habits. For example, after reading a book 83% of the readers check what the else the authors written and 75% look for similar books. [72] This shows that displaying other books by the same author and similar books may be valuable for users. The leading number of users, almost 30%, said that recommendations by a trusted friend were the main reason for them to read a book. Reviews from Goodreads and Amazon influenced only 15% and 5% of users. Based on this data, friends’ recommendations might be another valuable functionality for the Booklio application.

After brainstorming and writing down the list of possible features, I divided the list into two groups: primary and secondary features.

Primary features:

- Searching for books by title, author or ISBN
- Adding/removing books to/from your library
- Adding books that you have read or wish to read
- Viewing book information, rating and reviews

Secondary features:

- Viewing book recommendations based on books you have read
- Rating books and writing book reviews
- Viewing personal reading metrics by month and year
- Adding other users as friends and viewing their libraries
- Recommending books to friends
- Requesting to borrow friends' books and tracking lent books
- Searching for books by scanning a barcode

Target Devices

Creating a mobile web app that is supported on a lot of platforms and browsers is a big challenge. The target devices chosen for this project include the iPhone and the iPad as well as Android smartphones and tablets. The application has also been tested on the Windows Phone, but no additional work has been done for optimizing the app for the Windows platform. Feature phones have not been included in the target group due to their small screens and limited browser support.

Content Design

Traditionally content development may not have started until a site was built, and may have been a part of the final validation and testing phases of the project. When developing a scalable user interface understanding the content becomes more

important than ever. The difference between the desktop and smartphone is screen size and real estate. Prioritizing what needs to be presented to the user in the mobile realm will help avoid overly long content pages or the need to break a piece of content into multiple pages, something which means longer download times and potential user attrition due to impatience.

4.3 User Interface Design

I have chosen a responsive approach with focus on a mobile first strategy for implementing the user interface. Marcotte states that responsive design implies choosing a reference resolution and progressively expanding it further for other contexts using the media queries. [46, 123] Starting with a mobile context seems like a very reasonable method as it helps the developer to focus on the essential elements of the application's layout.

A recommended practice for designing responsive web application or website is to use a user-centric information architecture approach. Such a user-centred approach means concentrating on the user's goals.

When developing a native app for a particular platform, developers must follow that platform's design guidelines, which specify the placement of navigation controls, user interactions and provide other user interface specifications. [4, 73] For example some UI functionalities might perform the same task but look differently on iOS and Android platforms. Designing a cross-platform web app raises the question of whether one should or shouldn't follow such guidelines. Creating an interface that adapts to each platform's specific guidelines would require a lot of additional work and code. Also different interfaces would have to be made for the desktop users.

Some opponents of mobile apps point out that mobile users expect an app to work similarly to other apps on their device's platform. However there is nothing preventing apps from looking similar on different platforms, but in this case it should be made clear to the user that an app is not trying to be a native application. For instance, it can be very confusing for a user if an Android app, which UI resembles the app made for iOS.

Since the main goal of the project is to make an application that can be used on different platforms and that works through a browser, there is no critical need for an application to look different on each platform. Platform guidelines apply mostly to native or hybrid applications and mention UI components and interactions present only in those types of apps. However, platform guidelines can still be also applied when designing mobile web apps. [4, 73]

Application Structure

Figure 23 shows the overall structure and general hierarchy of the application. The boxes in green colour outline the primary features of the application implemented in the prototype.

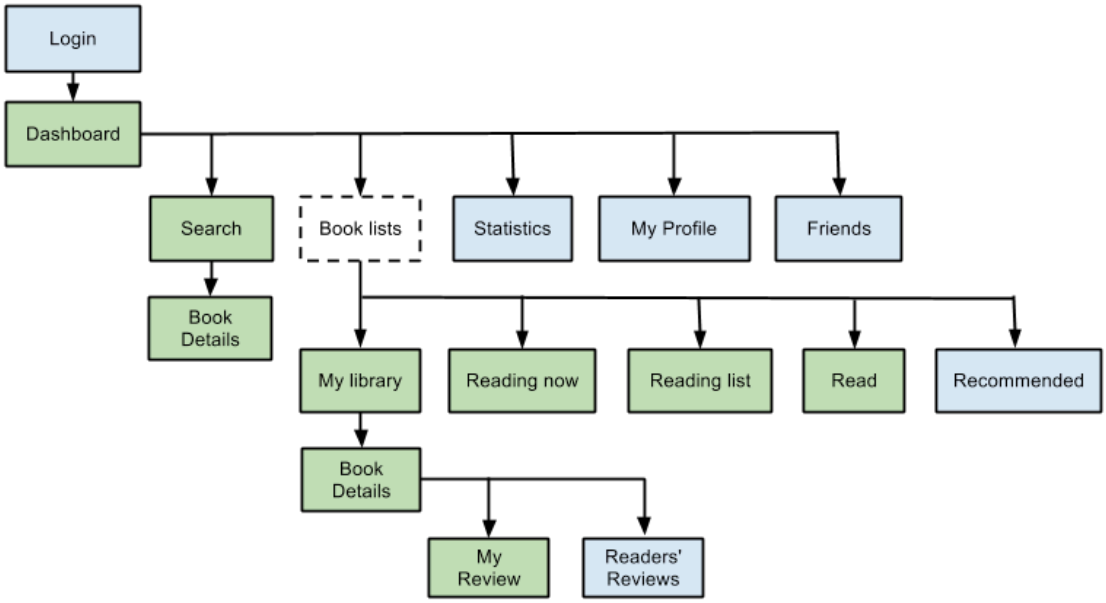


Figure 23. Booklio application map. Author's own visualisation.

Wireframing

Once the primary goals and functionalities of the app have been identified we need to plan the structure of the content. Designing for cross-platform and responsive layouts requires changes in the design process to accommodate for different contexts. Therefore, the first step is to outline the content without thinking about the final

layout. This will help to understand how the interface should communicate with the user, identify the features and plan the content for different contexts. Wireframing is an easy way to visualise the application's structure and flow and to test different ideas when designing the interface.

In this project I started the work by creating low-fidelity wireframes for small and large screens to get an understanding how the application might look and function. Figure Appendix 3 shows wireframes I have created to explore the initial UI ideas.

When designing a layout for multiple screens it is recommended to plan in advance how the layout will adapt to different screen sizes and orientations. [72] In the beginning of the design process, I started to realize that the wireframes do not provide enough flexibility to understand and test how the application's layout could scale. Therefore after wireframing the initial concepts, I continued the work by coding the HTML mockups using the responsive grid. That way I could actually try out resizing the layout in the browser and seeing how different UI elements scale and what adjustments need to be made. Appendix 4 shows one of the HTML prototypes that I have created used Zurb's Foundation[3] prototyping framework.

Structuring the Content

Readability research has shown that reading from a mobile screen makes it 108% more difficult to understand the content mainly due to limited visibility. [26, 102] Application content has to be adjusted in order for the layout to work on different screen sizes. Android user interface guidelines recommend utilizing multi-pane layouts to organize the view. [73] These patterns could be useful when designing a scalable layout also for a web app. For example, when viewing details for an item in the list, the user has to select an item and then details will be shown in the separate view (Figure 24). To return to the list view the user will have to press a back link or a return button.

3 http://foundation.zurb.com/

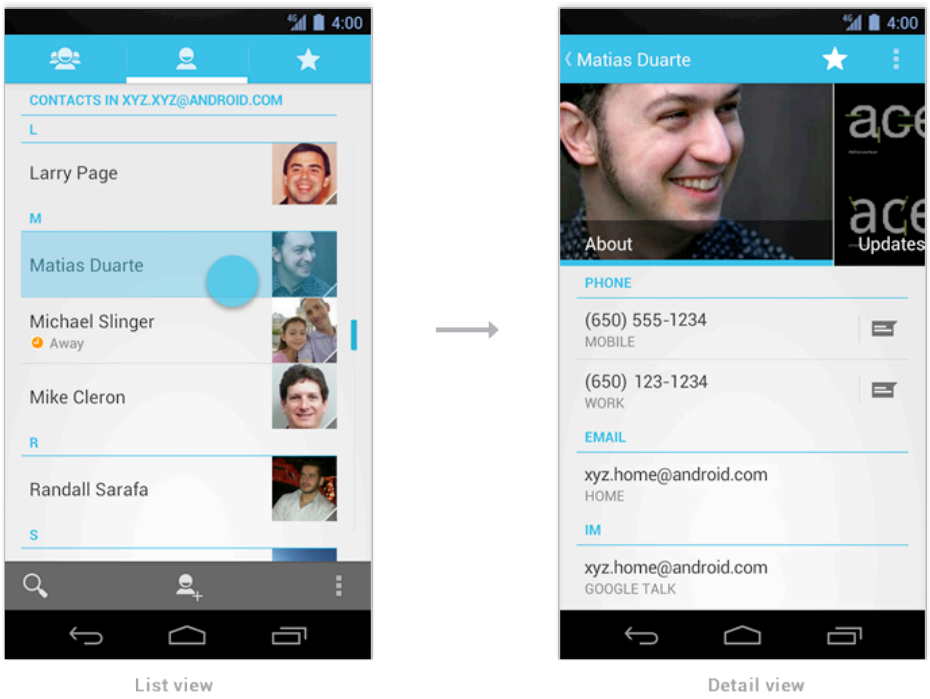


Figure 24. List and detailed views on a small screen. Copied from [73]

However, if there is enough horizontal space, for example when viewing the application on a tablet, Android user interface guidelines recommend combining multiple views into one compound view to utilize the screen estate more efficiently and simplifying the navigation as shown in Figure 25.

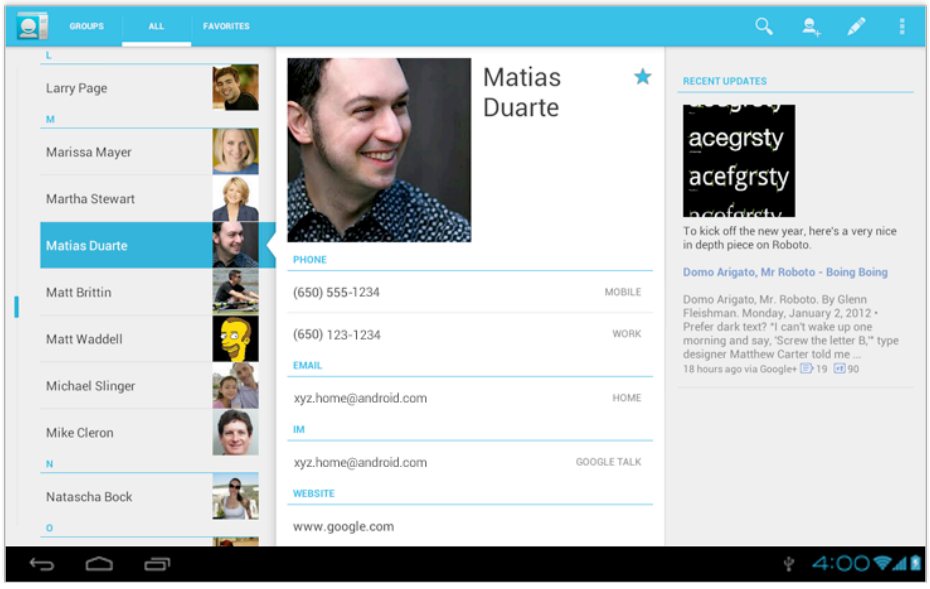


Figure 25. List and detailed views combined in one compound view on the larger screens. Copied from [73].

I found that in Booklio this pattern could work very well to for example display search results and detailed information for each result on medium and large screens (Figure 27, Figure 27).

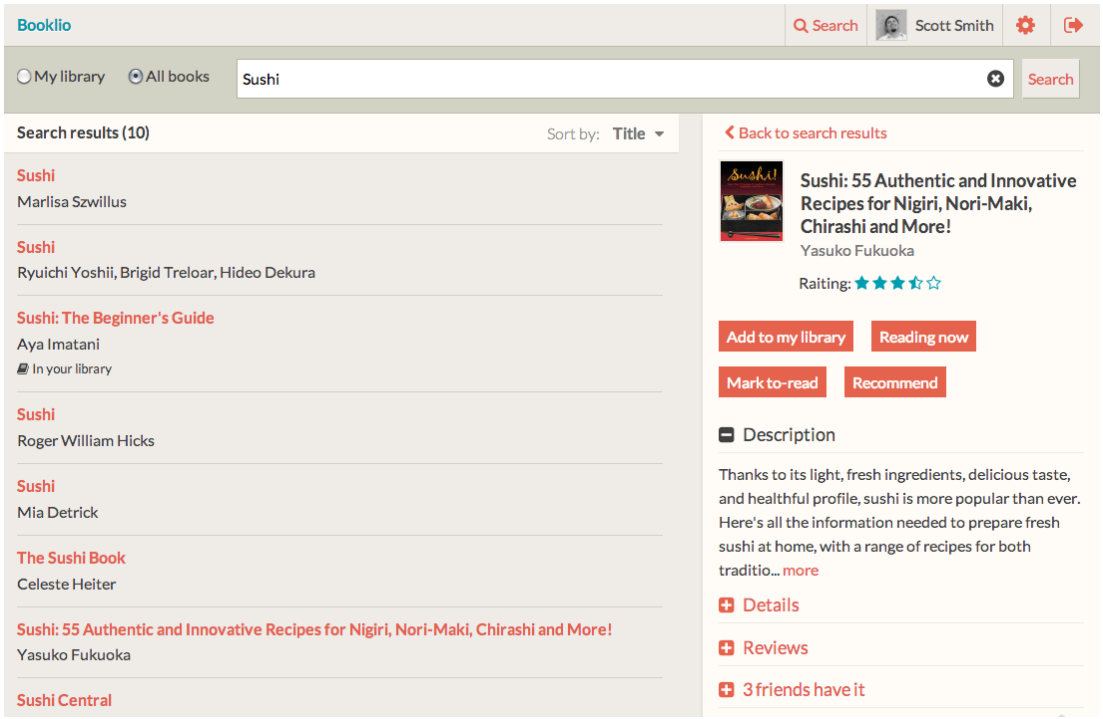


Figure 26. Search results and details view in Booklio on a medium-sized screen. Author's own visualisation

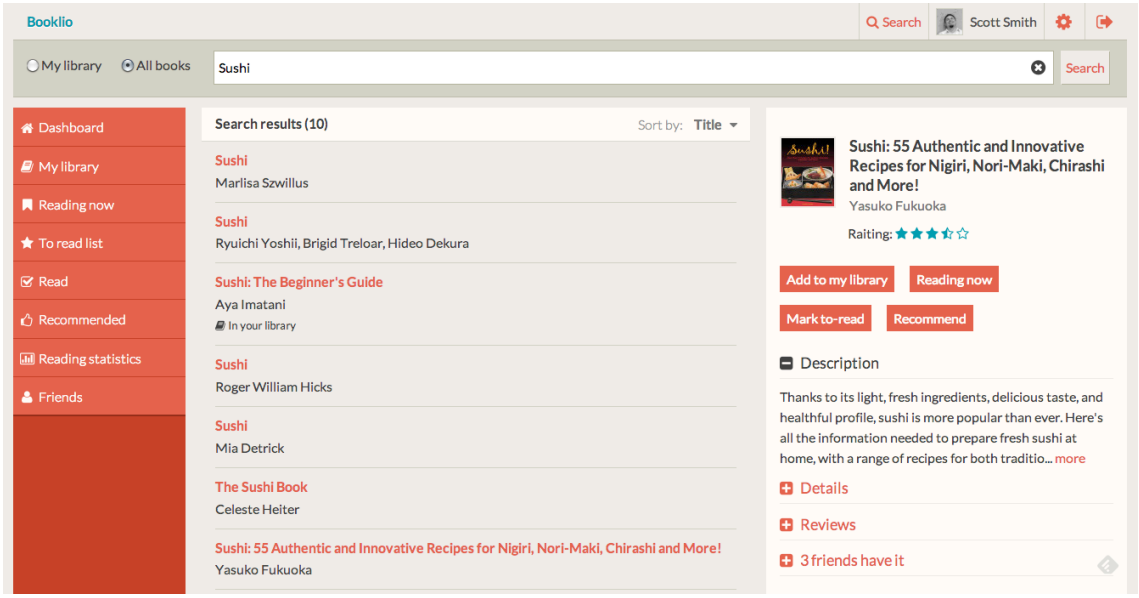


Figure 27. Search results and details view in Booklio on a large screen. Author's own visualisation

Another recommendation is to strive for functional parity on different orientations. [73] For example, it is preferable not to split up the compound views on the orientation changes. Parity can be achieved by adjusting the content column widths, stacking the panels or collapsing the left pane view to show only the necessary information.

Over the few years of the responsive web sites development a number of patterns for adapting multicolumn layouts to different screen sizes have emerged. [74] In some patterns content columns become narrower and stack one below another only on the smallest screens, in others, columns are moved below others as the screen width decreases. There is also a pattern called ”off canvas”, which instead of stacking columns, uses space off the screen to keep the extra column hidden until the screen width is bigger or until the user chooses to view the hidden column (Figure 28).

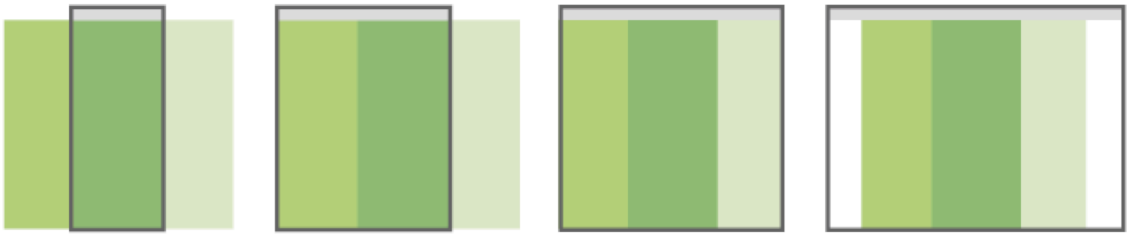


Figure 28. Off canvas layout pattern. Copied from [74].

This pattern seems most suitable for organizing the layout in Booklio, since the application’s layout consists of three columns and some columns such as navigation and details view would work best on smaller screens if they can be viewed besides the main content and not below to prevent unnecessary scrolling (Figure 29).

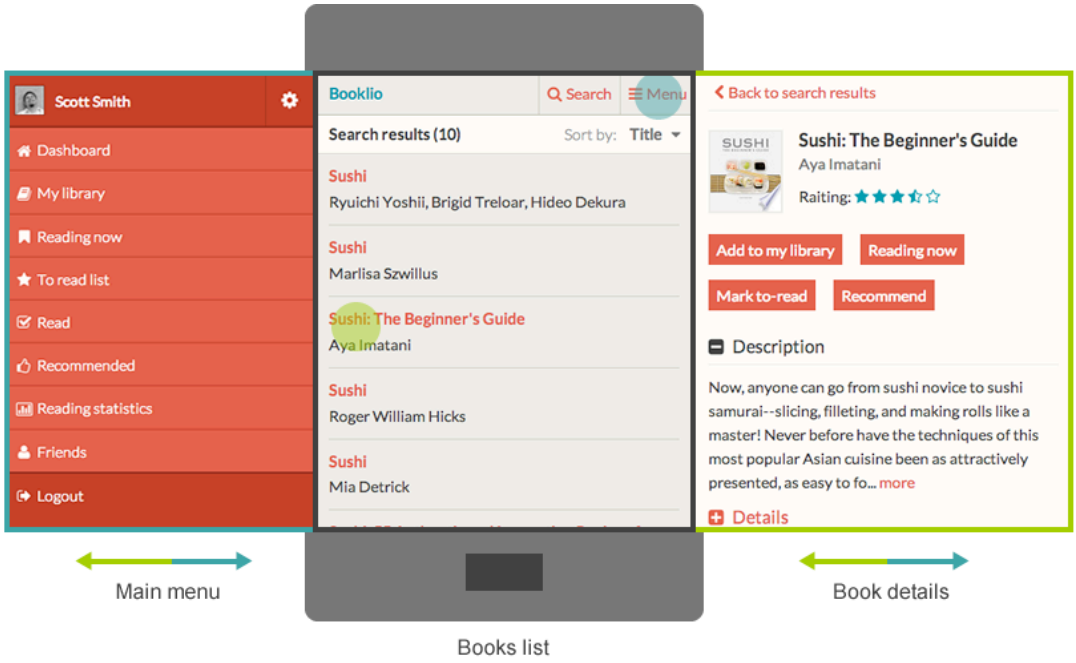


Figure 29. Off canvas layout pattern in Booklio. Author's own visualisation.

Navigation

Navigation is another critical element for the app's accessibility and usability. App navigation in Booklio is divided into three main categories as categorized by Tidwell [75, 80]:

- Global navigation contains the links to the content sections.
- Utility navigation with links to sign in, sign out, profile, settings.
- Inline navigation presents options related to the current content.

On large screens there is enough screen space to make navigation always visible and easily accessible. According to the established website conventions global navigation is usually placed at the top or left side of the page. If a page is built using a fluid layout, navigation can be also placed on the right since horizontal scrolling will not be an issue. [75, 85] Utility navigation is commonly found in the upper right part of the page.

However, on smaller screens and on devices with touch support more work is needed for planning the navigation system. Navigation then has to fulfil the following requirements [75, 85]:

- It should occupy a minimum amount of screen space and leave more focus for the main content.
- It should be intuitive and easily accessible when needed.
- It should be usable by a wide variety of devices.

For mobile apps it is important to show its top-level navigation structure and this is usually achieved by displaying a persistent menu bar at the top or bottom of the screen. [75, 448] As described in Chapter 2.18, thumb zones located at the bottom of the screen is the most comfortable area for touch interactions with one hand. This makes bottom of the screen a much more suitable area for placing the important controls and goes against the common placement of navigation controls at the top on the desktop web. The placement of the controls at the bottom prevents users from covering content with their fingers. The top part of the mobile view before the fold is the most important screen area and therefore a maximum of three crucial navigation links should be shown there. [75, 457]

iOS app guidelines recommend positioning the main controls at the bottom. However, Android devices have system controls at the bottom and therefore the Android guidelines suggest doing the opposite, i.e. placing the application controls at the top of the screen to avoid the accidental triggering of system controls.

Uneven support of ”position:fixed” CSS property across mobile browsers makes it difficult to create a navigation that is always fixed at the top or bottom of screen. [2, 172] One approach is to place the navigation at the bottom and include an anchor link at the top to jump to the navigation[4]. This way there is enough space for the content and navigation can be easily reached. On the downside, jumping to the footer might be slightly disorienting. The interaction itself is not as smooth as it is in the native apps

4 http://bradfrostweb.com/blog/web/responsive-nav-patterns/

where toggle or left flyout approaches are more commonly used. Also, if the navigation list is long it might look slightly awkward on pages with short content.

Another approach, instead of triggering a jump down to the bottom page, is a slide-down menu in the header. The benefit is that the menu then appears in the same place, there is no awkward jump and it is easy to scale it for larger screens. The drawbacks are that this does not make for very smooth transitions on some Android devices and results in dependency on JavaScript. A variation of this navigation pattern is a left-hand flyout menu. Clicking the menu link then opens the navigation menu from the left side and pushes the main content to the right.

For Booklio I have implemented left flyout navigation as it sits naturally with the chosen off-canvas layout pattern. Keeping the navigation on the left side also makes it more consistent across different screen sizes. Figure 30 shows the global navigation on smaller screens that is activated by selecting the menu link in the top right corner.

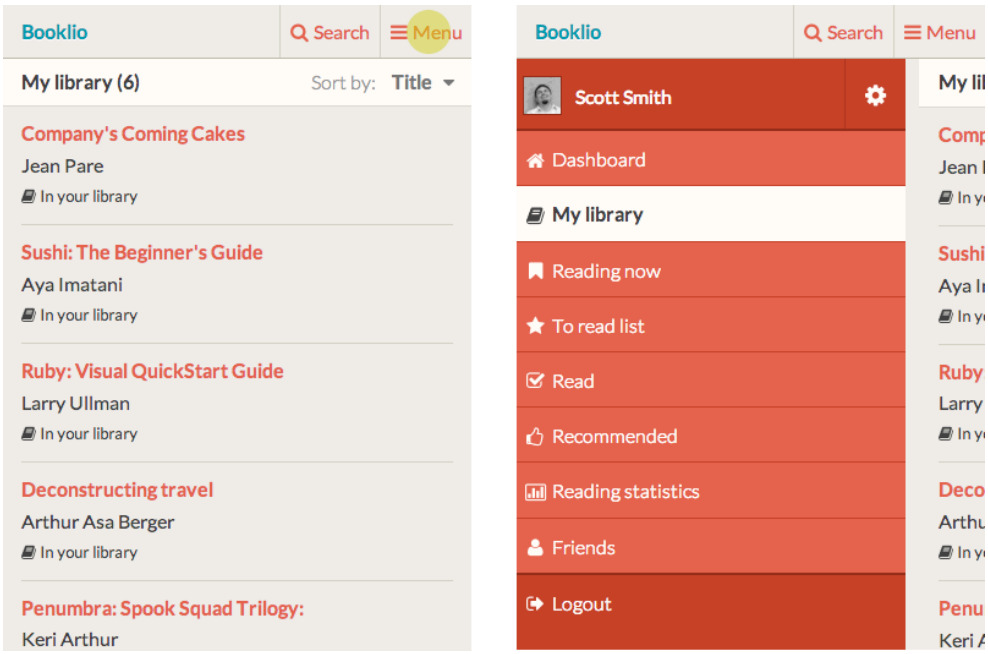


Figure 30. Global navigation is partially combined with utility navigation on small screens. Author's own visualisation.

On smaller screens utility navigation containing links to the user's profile and the logout link has been combined with the global navigation due to the limited space in the page header area. On medium and large screens the utility navigation is shown separately in the top right part of the layout (Figure 31).

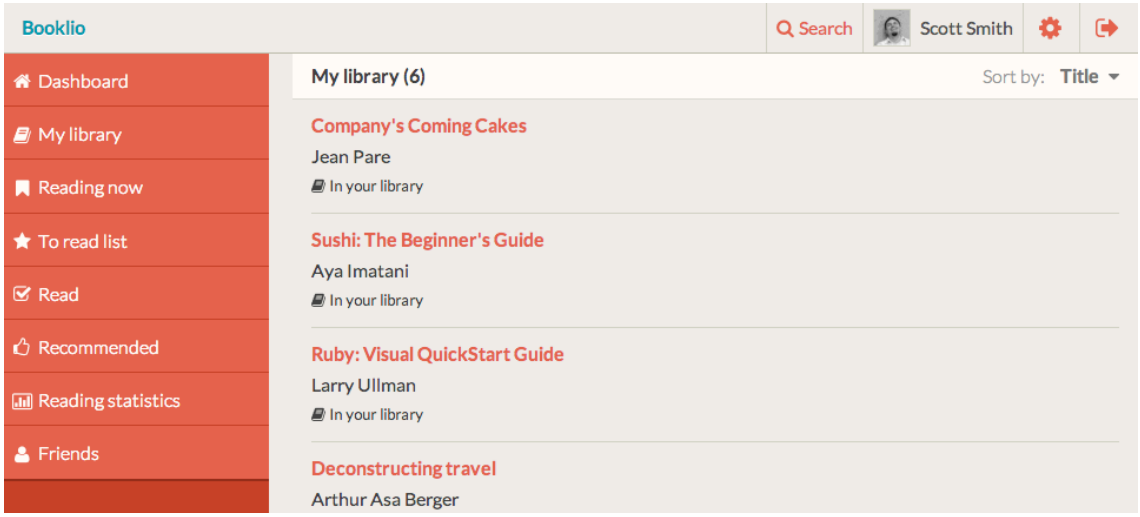


Figure 31. Global (left side) and utility (top right) navigation on large screens. Author's own visualisation.

The search function has been moved behind its own button. Initially I was planning to place the search field in the page header, but on the smaller screens the header space was limited. Also, the search function should allow the user to change the search category, so controls for that should be displayed as well. Therefore, I decided to place the search in its own block. Tapping the "Search" button will show the search field along with options to select the search category as shown in Figure 32.

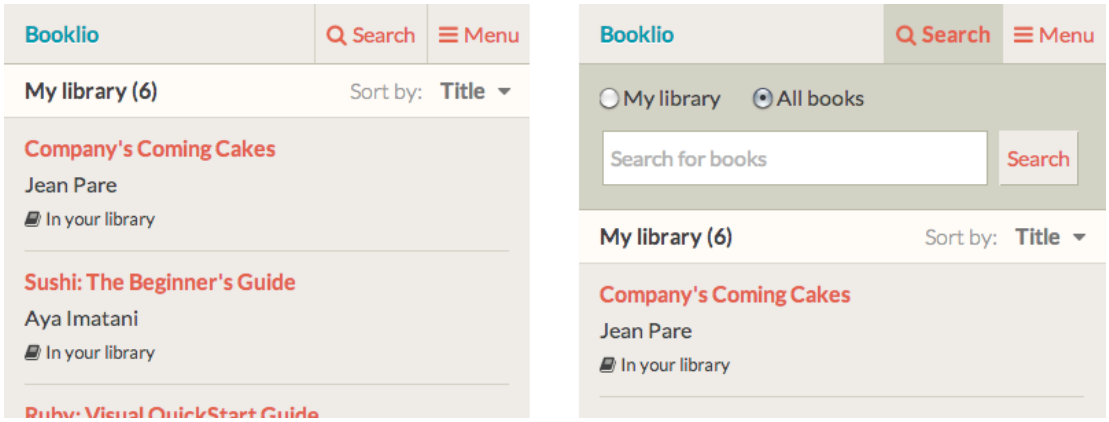


Figure 32. Search on smaller screens in Booklio. Author's own visualisation.

Most of the content in the application is in the form of book lists. For example, on smaller screens a list of all the user's books is shown as a selectable text list containing book titles and author names. On larger screens the same list is presented as a grid with thumbnail images of book covers and book ratings.

During the design stage I have been asking some possible users to provide a feedback on the mockups of the user interface. I also used Zurb's Verify[5] tool to collect feedback on the some of the mockups in order to decide how some views should look like, for example books listing on the large screens (appendix 5).

4.4 Application Architecture

The Booklio app consists of two main components: a client application that runs on the user's device and a backend application for handling data logic. Figure 33 illustrates the overall architecture of the Booklio app. Due to the scope of the project, Booklio does not include an authentication and user management systems, as it would have been in the real-life application.

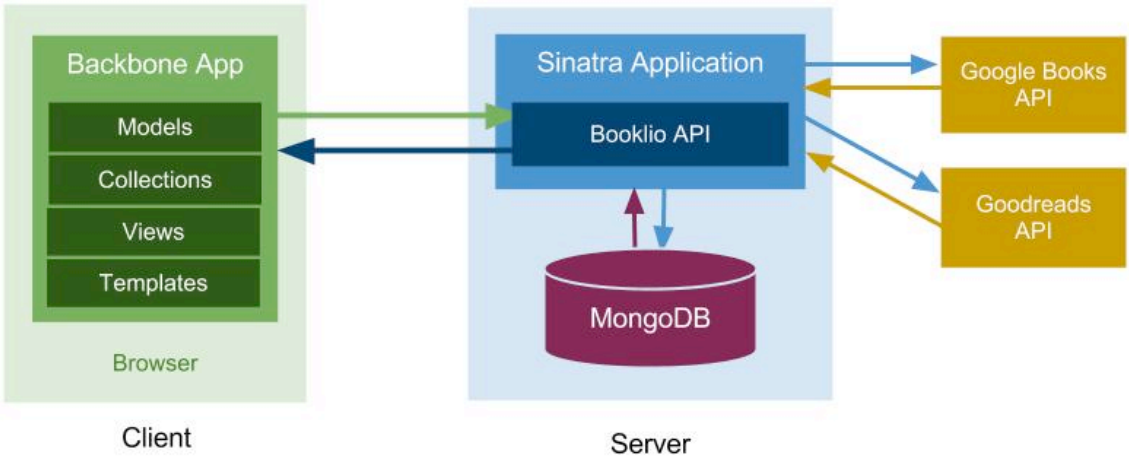


Figure 33. Booklio application architecture. Author's own visualisation.

5 http://verifyapp.com

Client side

The Booklio web app has been built using a single-page approach, which is one of the common programming models for the mobile web. Single-page apps contain the entire HTML markup in one HTML page and the primary benefit of this method is a better performance due to a fewer initial HTTP requests. [43, 46-47] Booklio consists of a single HTML5 index file that contains a base markup for including CSS files and a script that uses RequireJS to load the actual Backbone application including views and templates (lsiting 4).

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <title>Booklio</title>
        <link rel="stylesheet" href="assets/application.css">
    </head>
    <body>
        <div id="app" class="container">
            <header class="app-header"></header>
            <nav id="nav" class="app-nav"></nav>
            <div id="body">Loading Booklio...</div>
        </div>
       <script data-main="scripts/config"
       src="scripts/libs/require.js"></script>
    </body>
</html>
```

Listing 4. Example workflow in Yeoman for creating a Backbone application

The client side of the application is powered by Backbone.js[6], which is a minimal open source MV framework for writing HTML5 web and mobile applications. [58] It has a proven production usage and has been employed by many popular web services such as LinkedIn, SoundCloud and Foursquare. Backbone brings a structure to the web applications by providing models with key-value binding, collections and views. It

6 http://backbonejs.org

includes four components: Model, Collection, View and Router. All the views that observe the model can be notified of the changes and thus update themselves.

Often when developing JavaScript apps the code base becomes large and hard to maintain. One of the solutions to this problem is RequireJS[7], which is a JavaScript scripts and module loader based on Asynchronous Module Definition (AMD) pattern. RequireJS loads scripts asynchronously and only when needed, thus reducing application's load time. [76] Booklio's JavaScript code has been organised into modules using RequireJS.

Additionally, during the imlpementation I have utilised Yeoman[8], which provides a collection of tools for web development. One of the issues of current web app development is the too wide choice of best practices and tools. Yeoman helps to address this issue by providing tools such as scaffolding, making builds and package management. An example workflow for creating a Backbone app with Yeoman, accomplished just by running several commands, is shown in listing 5.

```
// Create a scaffold for a Backbone web app
yeoman init backbone
// Create Backbone view, model, collection and router using Yeoman's
// generators
yeoman init backbone:view bookView
yeoman init backbone:model bookModel
yeoman init backbone:collection bookCollection
yeoman init backbone:router bookRouter
// Install jquery library package using package management tool Bower
yeoman install jquery
// Build the application for deployment
yeoman build
```

Listing 5. Example workflow in Yeoman for creating a Backbone application

7 http://requirejs.org

8 http://yeoman.io

The build task function is used to construct an optimised version of an application that is ready for deployment. Its automated integration with the application helps to speed up the development. In this project, the build function has been performed each time before deployment.

Other libraries used in Booklio include:

- Requre.js for AMD module and script loading support
- jQuery for DOM manipulation
- Underscore.js for templating
- Modernizr for feature detection
- Hammer.js for touch-based interactions support

Server Side

The server side of the application consists of an API that handles data retrieving and storing. The API is served by a Sinatra application running together with a MongoDB database for data storage. The API is also responsible for fetching book information, such as cover images, author details and book descriptions, from the Google Books API. The Google Books API does not provide any book reviews or ratings information. Therefore, this information has to be fetched from the Goodreads API. Due to the scope of this project I will not go into further details on how the server side was implemented.

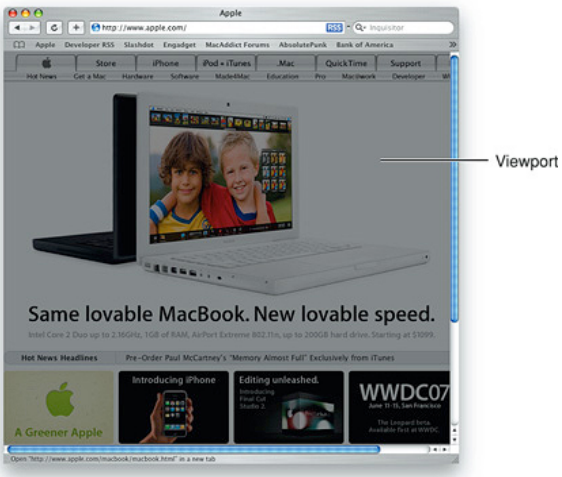
4.5 UI Implementation

Booklio's interface was built using a semantic HTML5 markup, CSS styles and JavaScript. Semantic HTML5 markup keeps adaptive experiences manageable and accessible, and also makes it easier to progressively enhance the user experiences. For example, using proper HTML5 input types brings an appropriate virtual keyboard on many touch devices, for example containing only numbers for phone number input. Additionally, semantic markup adds the benefit of portability and can be accessed by many mobile devices, tablets, desktop browsers and other web-enabled devices, regardless of the feature set or capability.

Setting the Viewport

Pixels visible in browsers are not the same and divided into device and CSS pixels. The difference between them is that device pixels mean an actual pixel size of the screen and CSS pixels refer only to the visible area in the browser window. [21, 58] In practice this means that on some screens CSS pixel is equal to a device pixel, on other screens, such as high-resolution Retina displays in iPad, CSS pixel is equal to two device pixels. [8] When a user zooms a page in or out, CSS pixels stretch or shrink and their visible number changes, but the amount of device pixels remains the same.

These pixel differences are important to keep in mind when configuring a viewport for mobile usage in order to avoid involuntarily changes in webpage's zoom level caused by the mobile devices. [4, 125] Due to content interaction differences on mobile devices, the viewport on a desktop is slightly different from the viewport for example on a smartphone. As is shown in figure 35 "desktop viewport" refers to the browser's visible area, but on mobile it can be larger or smaller than the visible area. [21, 58] In other words, this is the area that determines how the content is presented and wrapped on the webpage. The mobile viewport can be separated into two parts: the layout and visual viewports (Figure 34). [21, 58]



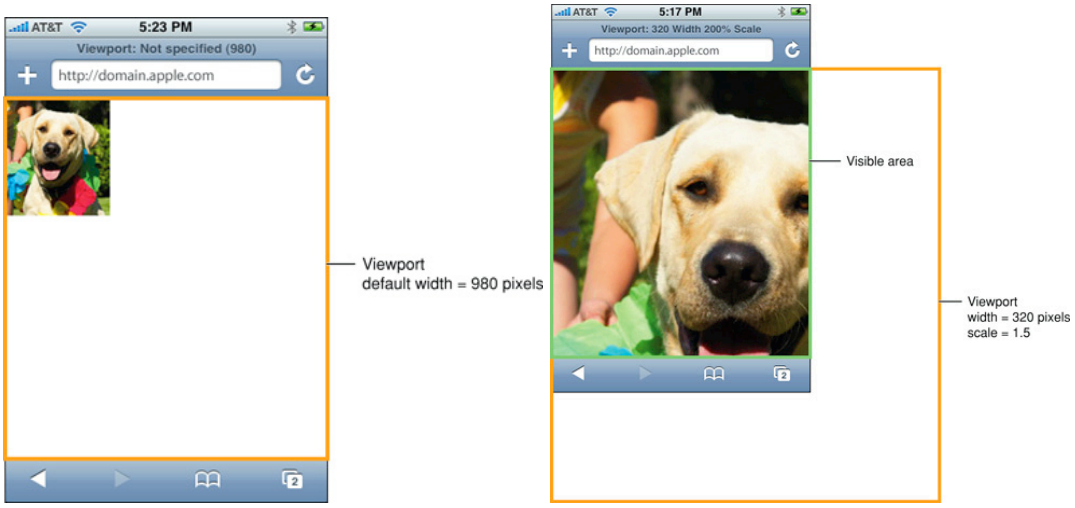


Figure 34. Viewport differences on a desktop and mobile browsers. Copied from [77].

The layout viewport is the same as on a desktop, and represents the actual size of a webpage. It is similar to device pixels as it retains the same size, regardless of zoom level. The visible part of the page refers to the visual viewport and its size can be changed by zooming or scrolling to different parts of the page. [21, 58; 66] Mobile browsers need to set the default viewport to the layout viewport if it is larger than the visual viewport.

To improve the presentation of web content, Apple recommends setting the viewport size and defaulting zoom level using the ”viewport” meta tag. [66] Apple introduced this tag in the Mobile Safari browser to let developers control the viewport’s scale and size and support for the tag has been extended to other browsers[9]. For mobile layouts some of most common approaches are to set the viewport width parameter, *width*, to ”device width” and either set both minimum and maximum zoom level set to 1.0 or only initial zoom of the page and maximum level, as shown in listings 6 and 7.

```
<meta name="viewport" content="width=device-width,minimum-
scale=1.0,maximum-scale=1.0"/>
```

Listing 6. Configuring viewport settings with a fixed size

9 https://developer.mozilla.org/en-US/docs/Mobile/Viewport_meta_tag

```
<meta name="viewport" content="width=device-width, initial-scale=1
maximum-scale=1">
```

Listing 7. Configuring viewport settings to prevent zooming in on a webpage when orientation changes.

Figure 35 shows how various values of *device-width* setting affect the size of the layout elements in the browser.

Figure 35. Viewing 300x300 pixel image in the default Safari viewport, in a 1500-pixel viewport, at a device-width set to the scale of 1.0 and at device-width set to scale 2.0. Copied from [4, 127].

Viewport tag example shown in listing 6, tells the browser to make the viewport's width the same as the device's physical width. Setting ”initial-scale=1” and ”maximum-scale=1” ensures that a browser does not try to change a zoom level when a device's orientation changes. However, one downside to this is that the ”maximum-scale=1” property disables user manual zooming, which is a behaviour users are accustomed to on mobile phones and this option therefore decreases the users experience. Without the ”maximum-scale=1” property, users have to pinch zoom to get the same text size after changing the orientation. Preventing user scaling is nonetheless a much bigger drawback, so in this case omitting ”maximum-scale=1” property is a better solution.

Choosing the Layout Breakpoints

An important step in creating a scalable layout by using responsive grids is to identify the layout breakpoints, i.e. the horizontal widths needed for accommodating the

responsive design. [53, 113] Based on my research, there are a few different approaches to this. The question of which approach to choose depends on the case in question as there is no standard way for this process.

One of the popular approaches is to define breakpoints based on common screen sizes, for example, 320px for smartphones, 768px for tablets and 1024px for desktop computers. However, this might lead to a risk of supporting specific screen sizes and ignoring the in-between screens. Since there are devices with all kinds of different screen widths, finding common screen sizes might become difficult. Changing device orientation adds additional challenge.

According to Kadlec a better approach is to let the content define the breakpoints by checking where the layout breaks when the screen size changes. [21, 79] By resizing the browser window to the minimum width of 300px and then expanding it, content can be checked how it behaves at different window widths and when it needs to be fixed. This seems to be a better method since this way the content is prioritized and not the screen size. But, the problem comes from how to determine which way the layout is broken.

Due to a number of reasons, such as varied zoom factors when browsing, different default font sizes on devices; defining breakpoints in pixels might not be the most optimal approach. Using pixels to specify the breakpoints might cause the layout to break or content widths to become off in some situations and it is hard to predict when that might happen as the web is very unpredictable. [21, 85] Using em units for the breakpoints and defining measurement for other elements can lead to a much more flexible solution that is compatible with the big variance caused by screen size, pixel density and zoom level and is generally more future friendly. The term "future friendly" refers to the support of new devices whose size and specifications might be unknown. [21, 102]

An em unit is equal to the current font size, for example 1em is 16px and is resizable across browsers. [20, 29] The pixel-based value can be converted into ems by dividing the target by the context, where target in this case is a breakpoint and the context is a

body font size. [20, 86] Using the media queries in em units ensures that the layout will be scaled properly when resized or zoomed in.

Nowadays, there are over 20 frameworks that provide fluid CSS grids for responsive layouts, for example Zurb Foundation, Twitter Bootstrap and Skeleton. However, most of the fluid frameworks come with predefined grids and styles for the UI elements. Often developers will have to build the app into a framework's grid and override the existing styles, which might be not ideal for creating custom UIs and will cause unnecessary overhead.

In this project I have chosen to use the Compass CSS[10] framework that is based on the extension of CSS3 called SASS. SASS lets developers use nested rules, variables, functions and utilities, and helps to organize and maintain stylesheets as well as generates well-formatted CSS. Compass provides a number of useful reusable patterns and plugins for simplifying the CSS development, but it does not provide any default styles. Compass does not come with a grid system either, but it is possible to create one's own grid system by using mixins. Another alternative is to integrate another CSS framework that has grids.

I have chosen a responsive grid plugin called Susy[11], that provides a number of grid helper functions, or mixins as they are referred to in SASS terms, for creating one's own scalable CSS grids in a semantic-friendly way. Susy works by putting grid elements in a row one after another by positioning each element in relation to its nearest neighbour. The last element, 'omega', is floated to the opposite side of other elements in the row.

The basic Susy grid utilizes just two mixins:

- container() for establishing the initial grid context.
- span-columns() for declaring the width of the grid elements.

10 http://compass-style.org/

11 http://susy.oddbird.net/

Listing 8 shows how a basic layout that can be created by using these mixins:

```
.page {
  // page acts as a container for our grid.
  @include container;
  // nav spans 3 columns of total 12.
  nav { @include span-columns(3,12); }
  .content {
    // content spans the final (omega) 9 columns of 12.
    @include span-columns(9 omega,12);
    // main content spans 6 of those 9 columns.
    .main { @include span-columns(6,9); }
    // details content spans the final 3 (omega) of 9 columns.
    .details { @include span-columns(3 omega,9); }
  }
}
```

Listing 8: Basic layout using Susy grid mixins. Taken from http://susy.oddbird.net/guides/getting-started/#start-responsive.

When starting with Susy, a few variables need to be set for calculating the correct element widths: i.e. the default number of columns, column widths and the grid gutter (listing 9).

```
// Default number of columns
$total-columns: 7;
// Width of each column
$column-width: 5em;
// Space between columns
$gutter-width: 1em;
// Space on the right and left of the grid
$grid-padding: $gutter-width;
// Breakpoint for the medium-sized screens (tablet)
$medium-columns: 8;
// Breakpoint for the larger screens (desktop)
$large-columns: 55em 12;
```

Listing 9. Default Susy settings for the grid layout used in Booklio.

The mobile first approach was used in this project and therefore the *$total-columns* variable refers to the number of columns that will visible on the smallest screens. For larger screens, such as tablet and desktop, additional breakpoints have been specified: *$medium-columns* and *$large-columns*. The medium breakpoint can fit eight columns and is aimed mainly at tablet-sized screens. The large breakpoint will be triggered mostly on desktop screens with a width of over 55em and the layout will change to a 12-column grid. The 55em value was chosen after testing where the layout clipping started to appear on the tablet sized screens and trying different values to find the layout that was becoming too large for the medium sized screens. The values for these breakpoints have been chosen during the implementation stage by testing how the content fitted on the different-sized screens.

Susy's *at-breakpoint* mixin lets the developer easily specify the breakpoints. The required CSS rules can be written within each breakpoint scope as shown in listing 10.

```
@include at-breakpoint($medium-columns) {
  .app-nav {
    @include span-columns($side);
    margin-left: 0;
    .show-details & { margin-left: - 100%; }
  }
  .main {
    width: columns($main);
    .show-details & { margin-left: 0; }
    .show-menu & { margin-right: 0; }
  }
}
```

Listing 10. Susy mixin for medium breakpoint.

CSS implementation in Booklio has been done by first writing baseline shared styles that will be used in all views and later writing separate stylesheets for medium and large screens. This way the CSS structure could be kept simpler and more maintainable.

Backbone's views use Underscore.js templates, which are HTML files containing Embedded Ruby style markup (listing 11). Code wrapped in <%= %> tags denote a

variable that will be replaced after compilation with actual data. Templates can also include other standard JavaScript code, for example, if-statements.

```
<script type="text/template" id="book-template">
<div class="title"> <%= title %> </div>
<div class="author"> <%= author %> </div>
<% if(typeof(cover) !== “undefined”){ %>
  <div class="cover"><img src="<%= cover %>"></div>
<% } %>
</script>
```

Listing 11. Underscore.js template.

Handling Touch Events

Booklio uses progressive enhancement to add support for touch-enabled devices. Modernizr can help to find out most of a browser’s supported features, but some, like touch support, are more difficult to determine reliably. Modernizr has tests for touch support by checking for presence of the *touchstart* event, but since some browsers do not use this event even on touch-based devices, this approach would not always work. [2, 76]

One of the primary requirements for touch screens is to make large touch targets for links, buttons and other controls. [75, 465] Each mobile platform has its own recommendations for the best touch target size. Unfortunately there is no standard size for the comfortable minimum size of touch UI elements. For example iOS Human Interface guidelines recommend a target of 44px in width and height, whereas Windows Phone guidelines suggest using touch targets that are 34px wide and high. One of the ways working around the issue of diverse target sizes is to add enough inner margins and tappable whitespace around the element. [75, 465] This way there is no need to make buttons overly large, something that will also help to preserve the consistency in the layout when it is scaled.

One of the functionalities that could benefit from touch support is the opening of the main menu and details views by using a swipe gesture. To implement the swipe

gesture support I have used the Hammer.js[12] library that provides the possibility of enabling various touch gestures.

Adjustmenting Layout for Different Screen Sizes

For the smaller screens some content elements need to look differently than on the larger screens. While many adjustments can be achieved by using media queries and CSS, sometimes the elements would need a totally different markup and therefore require different view templates.

The obvious solution for loading templates based on the screen size is to check the width of the screen when the application is loaded and then load the appropriate template. However, this means that the breakpoint values would have to be also specified in the JavaScript in addition to the CSS.

Another option is to check the position of some of the elements, for example the main menu. Conditional loading can be made based on the position of the main menu element: if it's outside the main container, then there is a prompt to load the template for smaller screens, otherwise the app is prompted to load the template for larger screens.

Optimizing UI for Performance

CSS provides a wide range of options for creating a nice user interface, but some features can affect the performance on low-end devices. For example, some CSS properties like gradient, box-shadow and background-repeat use the device's GPU to paint the images on the fly. Therefore, GPU usage should be optimised by utilizing images more efficiently. One recommendation is to use image sprites so that the device downloads only a single image or to embed data URIs in the CSS files for smaller images. [43, 22]

12 https://github.com/EightMedia/hammer.js

Some of the properties that do not require repaints include *transition*, *opacity* and *transform*. In Booklio I used a *transition* property to create the sliding animations, for example, when opening and closing the global navigation on small screens (listing 12).

```
.app-nav, .details, .main {
  @include transition(.4s all ease);
  height: 20em;
}
```

Listing 12. Transition property applied to the global navigation, details and main elements.

By having a *transition* property applied to the navigation element and changing the position of the navigation, the navigation appears by sliding from left to right and vice versa.

In Booklio I tried to minimize the usage of images to limit the amount of downloaded assets and avoid creating images for different screen resolutions. Therefore, I have built the entire user interface using only CSS and used an icon font for the interface icons to reinforce the function behind the navigation elements. [2, 39] Icon fonts have the following advantages in comparison to plain images:

- They can be easily styled with CSS.
- They look good with various screen sizes and resolutions.
- Only one HTTP request is needed for loading all the icons.

The ”Font Awesome”[13] icon font was used to add simple monochrome icons to the Booklio UI. Monochrome icons also align well with iOS and Android app design guidelines, which also recommend using simple interface icons. [2, 39] Interface icons can be seen in figure 29 as they appear in the global navigation.

Appendix 6 shows the final prototype of the Booklio application on a smartphone (Samsung Galaxy SIII), a tablet (Samsung Galaxy Tab 2) and on a desktop browser (Chrome 27).

13 http://fortawesome.github.io/Font-Awesome/

4.6 Application Testing

Device Testing

The recommended testing strategy is to check that the application works in Firefox and then further testing should be made in Chrome and Safari. [17, 258] Also, testing can be done on emulators, but testing on the actual devices is essential as is suggested in W3C's mobile Web best practices guidelines. [78] An emulator might be the cheapest solution, but there might be some inconsistencies in browsers as compared to the real device and installing all emulators can easily bloat a developer's computer.

Fortunately, nowadays it has become easier to test on different devices with the emergence of services such as Keynote's DeviceAnywere[14] and Perfecto Mobile. Using these services, developers can access an array of mobile devices that these services provide and run their tests on the actual devices. During this project development stage I have used DeviceAnywhere to test my application (Figure 36). DeviceAnywhere provides access to a number of real devices in a shared system for web testing. Free version has a limited amount of available devices, but it is enough for getting a good overview of how an application behaves on a few key devices.

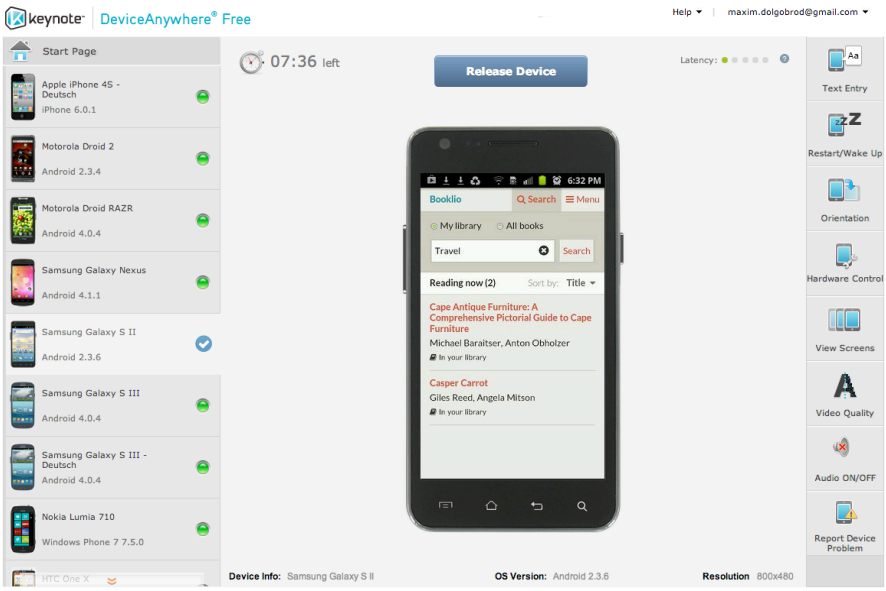


Figure 36. Device anywhere testing environment. Author's own visualisation.

14 http://www.keynotedeviceanywhere.com/

The DeviceAnywhere service worked very well for testing, however, there is a slight latency in the connection to the devices and sometimes it seemed that the device wasn't very responsive or fast. Figure 37 shows Booklio application testing on devices using DeviceAnywhere service.

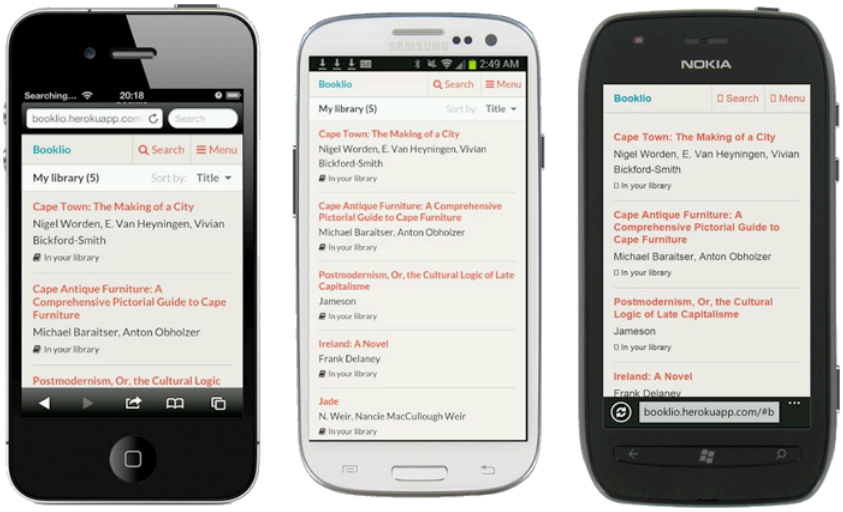


Figure 37. Testing Booklio on iPhone 4S (iOS 6.0.1), Samsung Galaxy S III (Android 4.0.4) and Nokia Lumia 710 (Windows Phone 7) using DeviceAnywhere service. Author's own visualisation.

During the development phase, Booklio has been frequently tested on actual devices running Android and iOS operating systems. Some tested devices included HTC Desire HD (Android 2.3), HTC Desire S (Android 2.3), iPod Touch (iOS 4) and iPad (first generation, iOS 5). Additionally, testing has been also made on a laptop and desktop computers using Chrome (version 26), Firefox (version 12) and Safari (version 5.1) browsers. During final testing I have evaluated how application was working on a device and whether the user interface was not broken and looked as it was designed to be. Appendix 7 contains a full list of tested devices, used during final testing, and found user interface issues.

Performance Testing

Booklio's performance has been analysed using Google PageSpeed Insights[15] tool. PageSpeed Insights is a free open source web page analysis service, which offers recommendations on how to optimize the performance of web pages. Appendices 8 and 9 contain the reports for mobile and desktop clients. Overall, the results were good, with only one high priority and seven low priority issues for the desktop clients. The mobile report showed only two high priority and eight low priority items.

Also, the mobile report suggested trying to use an application cache, which is one of the supported features of HTML5. Every web browser uses a built-in cache that stores recently visited web pages to load them faster on the next visit. An application cache improves a browser's default caching futher by prefetching web pages and assets and being able to make all cached resources available in offline mode. [79] Performance testing showed that there is still a room for improvement and that the application can be optmizied even more for better performance and speed.

15 https://developers.google.com/speed/pagespeed/

5 Results and Discussion

5.1 Summary of Results

During this project I have built a high-fidelity prototype application based on the primary features outlined in chapter 4.2. Application has been optimised for viewing on smartphones, tablets and desktop computers and includes support for touch gestures. Application allows to performs the following functions:

- Search for books, display their information and check if book is already in user's library
- Add books to an own library
- Mark books as to-read and currently-reading
- Rate books and write reviews
- View book reviews from other users

Due to the time constraints, no official usability testing was performed at the end of the project, but the application was tested on a number of smartphones, tablets and desktop computers. After designing and developing of the Booklio application, the following answers can be drawn to the research questions raised in chapter 1:

What are the main challenges in developing a cross-platform user interface without any cross-platform UI web frameworks?

One of the primary goals when developing a cross-platform user interface is to strive for creating reusable UI components. That, however, can be challenging, since the same component might work on some devices, but not on others.

Inconsistent CSS support on mobile browsers presents another big challenge for building cross-browser CSS styles and achieving a consistent look on mobile devices. For example, varying support for CSS fixed position property in mobile web browsers can affect how some UI elements are displayed. Testing the Booklio application on some Android and Windows Phone devices revealed several minor UI bugs that were

not found during testing on desktop browsers. Most of these bugs didn't affect the functionality of the application, but did slightly decrease the overall usage experience.

Another challenge is adapting the user interface to different screen sizes and resolutions while maintaining the visual and information continuity. This means that while layouts might differ depending on the screen size, UI elements should retain the same feel across the devices and users should be able to confidently find the information across the devices.

Finding a suitable JavaScript framework for implementing a user interface that fits with the application's scope can also present certain challenges. Building a prototype application in this project using only Backbone.js without additional user interface libraries, in the end resulted in a slightly large amount of view code, for example for making the nested views. In the end I came to conclusion that, while Backbone.js was a lightweight and easy to use framework, combining it with some Backbone extension library that already provides a support for nested views and layout management could simplify the user interface development.

One of such libraries is a Backbone.Marionette[16], which helps to reduce view code by providing specialised view types and brings application architecture to Backbone, including view management. While Booklio's user interface was not very complicated, I came to conclusion, that using a library like Backbone.Marionette would have simplified the implementation of the views.

What are the most important design considerations when developing a cross-platform UI?

The most important design considerations that can be concluded when developing an application of this kind are to start with a mobile first layout and continue gradually enhancing it bit by bit. Mobile first media queries provide a widely usable default for the layout that works on different device types. Using em-based breakpoints in CSS media queries allows achieving a scalable layout indepenedent on the device's size.

16 https://github.com/marionettejs/backbone.marionette

Not relying on comfortable assumptions that UI components will work the same way on across mobile browsers can help to design for a more adaptive experience.

Context usage should be considered and well understood in order to prioritise what information to display when there is limited screen space. It is preferable to focus on the content first in order to present the required information to users clearly and quickly.

Also, it is necessary to plan early for how the application layout will scale and how navigation and action controls will work on different screen sizes and with different types of input. Therefore, it is important to do as extensive prototyping as possible in the design phase to resolve all issues with UI scalability.

Is it feasible to achieve a consistent user experience both on mobile and desktop devices in a cross-platform web application without any cross-platform UI web frameworks?

In the end of this project I have tested Booklio prototype on 17 devices and 5 different platforms and user interface has performed and looked consistenly on almost all of the tested devices, except 5 of them (appendix 5). Also Booklio did not use any cross-platform UI web frameworks, such as jQuery Mobile or jQTouch. Therefore, I think it is feasible to achieve a consistent user experience in a cross-platform web application on both mobile and desktop devices using only custom developed UI components. However a more detailed user testing is needed to be able to evaluate the user experience from an actual end user point of view.

Depending on the degree of application complexity and need for native hardware access, it is possible to build a fairly consistent user experience in a cross-platform web application. Among the important steps to achieve the consistency include performing a strict feature prioritisation, early prototyping in the browser and testing on at least a few actual devices when developing a user interface.

5.2 Further Development

When developing the Booklio prototype application no dedicated fallback has been made for devices without any JavaScript support, since all target devices for the project have come with enabled JavaScript. Ideally, when developing a complete web app, adding support for devices with no JavaScript should also be considered. From the development point of view addressing the missing JavaScript support would require building the application to first work without JavaScript. In case of Booklio this would also mean a completely different solution for the client side, as usage of any JavaScript-based framework would be out of the question. The application would have to be build like a normal website, where each request would require a page reload.

Further usability improvements can be made by utilizing a device's camera to scan the book barcodes to assist in the book searches. This could help to eliminate unnecessary typing, especially on smartphones, and aligns with mobile interactions heuristics. [75, 445] Figure 38 shows a mockup of an interface that would support barcode scanning on the devices with enabled camera access.

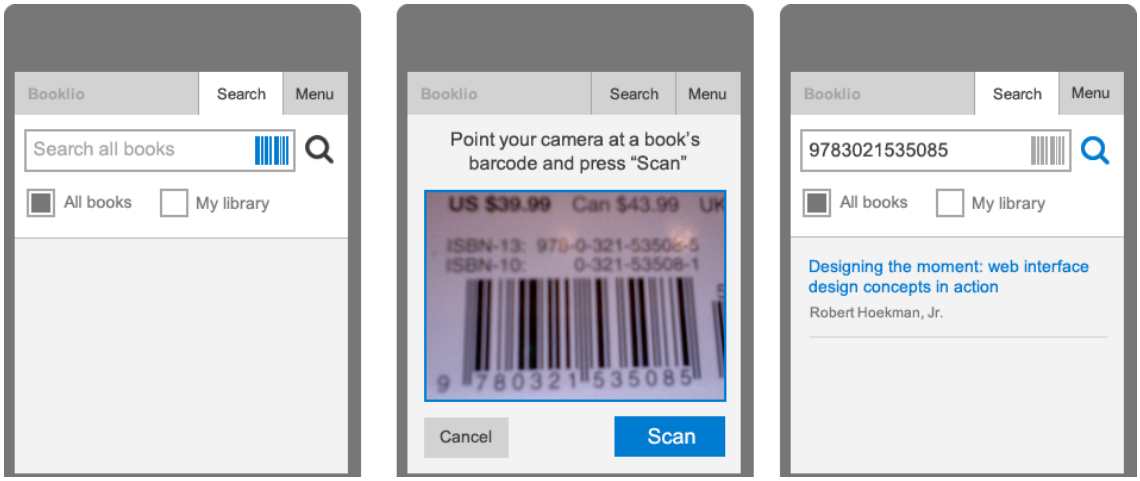


Figure 38. Barcode scanning in Booklio. Author's own visualisation.

Currently, only only a few desktop browsers support the getUserMedia API, which allows camera access. Therefore, this functionality will not be possible to implement for mobile devices[17]. Alternatively, voice input could be used with the support of the Web

17 http://caniuse.com/#feat=stream

Speech API. The user could simply say the title of the book to the device and the application would then complete a search based on the spoken text. However, while the Web Speech API allows adding speech recognition to web apps, it is only supported in Chrome's desktop version.

Cross-device usage patterns are becoming more and more common among users that use multiple devices. [74] The Booklio application could be made to support that behaviour. For example, the application could save the recent search history and user's last opened view. When the user then logs into Booklio on another device, the application could automatically redirect the user to the last visited view. Saving the search history would help to avoid repetitive typing. Of course, these improvements would have to be tested with real users and validated to see whether they are helpful for this sort of application.

Offline support could be another useful improvement. Based on discussions with potential users, one of the most important features in the Booklio application is the possibility of checking whether user owns or has read certain book. For example, if the user is travelling abroad and a data connection is unavailable, the user can nonetheless check an offline list to determine whether a book spotted in a book store is already in the user's library or not. Combining the offline data in local storage with HTML5 application cache could significantly improve application's user experience when there is no network connectivity.

6 Conclusions

This master thesis project has been initiated as a research on how to design and develop a user interface for cross-platform web application. I wanted to find out what are the main challenges and design considerations for creating the cross-platform interfaces based on the current web technologies and existing techniques. An additional task was to find out whether is it feasible to achieve an optimal usability in such interfaces. Finally, a limited functionality prototype of a cross-platform web application has been built and its user interface has been tested.

The prototype did not have all the desired features as I was planning originally and therefore not all UI features have been tested. Testing revealed certain issues, like some minor inconsistent CSS behaviour and data loading via AJAX on some platforms and more work would have been needed to resolve those. Also the developed prototype application was aimed at the high-end smartphones and would have required more optimisations to work on lower-end phones. However as a proof of the concept, the prototype performed well and I was able to test how the UI worked on different devices.

Even though more work is still needed for the UI to be ready for a production level release, the overall outcome was positive. With detailed planning, minimalistic approach and constant testing it is feasible to create cross-platform user interface for a web application. Consistency in user experience and inconsistent CSS support across platforms are among the some of the main challenges when creating a cross-platform user interface.

Cross-platform web application development is a challenging area and during the process it is very important to start with a simple structure, test often on the real devices and use responsive and progressive enhancement techniques to provide the user interface scalability.

Even though Nielsen admits that native apps perform much better compared to mobile websites, mobile web application might be a better strategy for the future [11, 35]. Future phones might not become significantly more powerful then they are now.

Combined together with rapidly evolving web technologies I believe that mobile web applications have a big potential in the near future.

The growing multi-device usage puts constant demands on information and services access anywhere and on any device. [74] Web browsers are evolving towards becoming additional application platforms. HTML5, CSS and JavaScript are widely used for creating web applications and we have been accustomed to use web apps on a daily basis. All this makes the web an already proven platform. New projects like Google's Chrome OS that uses the browser as a platform for running web applications and Mozilla's Boot 2 Gecko for mobile web applications push the boundaries towards a more open and unified web. [80] Also, HTML5 and the evolution of mobile devices have changed our perception of what is possible on the web.

References

1. Global Smartphone Shipment Reach a Record 700 Million Units in 2012 [online]. Strategy Analytics, Press Release, Boston, United Stated; January 2013. URL: http://blogs.strategyanalytics.com/WSS/post/2013/01/25/Global-Smartphone-Shipments-Reach-a-Record-700-Million-Units-in-2012.aspx. Accessed 10 February 2013.

2. Castledine E., Eftos M., Wheeler M. Build Mobile Websites and Apps for Smart Devices, SitePoint Pty. Ltd.; 2011.

3. Building Cross-Platform Apps with HTML5 [online]. Intel Developer Zone. URL: http://software.intel.com/en-us/articles/building-cross-platform-apps-with-html5. Accessed 17 May 2013.

4. Firtman M. Programming the Mobile Web. O'Reilly Media Inc.; 2010

5. Wilee H. Developer Wiki: XHTML MP [online]. Nokia Developer; August 2011. URL: http://www.developer.nokia.com/Community/Wiki/XHTML_MP. Accessed 10 December 2011.

6. Pranata A. Featured Freeware – Yahoo! Go 2.0 [online]. S60 Tips; January 2007. URL: http://www.s60tips.com/2007/01/12/featured-freeware-yahoo-go-20. Accessed 17 May 2013.

7. Ogg E. The living room PC is here: the iPad [online]. Gigaom; November 2011. URL: http://gigaom.com/2011/11/15/the-living-room-pc-is-here-the-ipad. Accessed 16 May 2013.

8. What makes a tablet a tablet? [online]. CNET; May 2010. URL: http://news.cnet.com/8301-31021_3-20006077-260.html. Accessed 16 May 2013.

9. Perenson M. Apple iPad Review: The Retina Display Redefines the Tablet [online]. PCWorld; March 2012. URL: http://www.pcworld.com/article/252010/apple_ipad_review_the_retina_display_redefines_the_tablet.html. Accessed 16 May 2013.

10. 1366 x 768 Most Popular Screen Resolution, Overtakes 1024 x 768: StatCounter [online]. TechPowerUp; April 2012. URL: http://www.techpowerup.com/164027/1366-x-768-most-popular-screen-resolution-overtakes-1024-x-768-statcounter.html. Accessed 16 May 2013.

11. Radwanick S., Aquino C. Mobile Future in Focus: Key Insights from 2011 and What They mean for the Coming Year. comScore; 2012

12. Cocotas A. Android Grabs A Record Share Of The Global Smartphone Market [online]. Business Insider, Australia; May 2013.
URL: http://au.businessinsider.com/
android-bounces-back-to-a-record-quarter-2013-5. Accessed 17 May 2013

13. Open Signal. The many faces of a little green robot [online]. Open Signal; August 2012.
URL: http://opensignal.com/reports/fragmentation.php. Accessed 6 February 2013.

14. Android. Screen Sizes and Densities [online]. Android Developers.
URL: http://developer.android.com/about/dashboards/index.html. Accessed 5 February 2013.

15. Android. Supporting Multiple Screen [online]. Android Developers.
URL: http://developer.android.com/guide/practices/screens_support.html. Accessed 5 February 2013.

16. Tablet App Quality Checklist [online]. Android Developers.
URL: http://developer.android.com/distribute/googleplay/quality/
tablet.html#optimize-layouts. Accessed: 10 October 2012.

17. Duffy T. Programming with Mobile Applications: Android, iOS, Course Technology, Cengage Learning; 2013.

18. Mynttinen I. The iOS Design Cheat Sheet [online] 2013.
URL: http://ivomynttinen.com/blog/the-ios-design-cheat-sheet-volume-2. Accessed 16 May 2013.

19. Microsoft. Multi-resolution apps for Windows [online]. Microsoft Dev Center; April 2013.
URL: http://msdn.microsoft.com/en-us/library/windowsphone/develop/
jj206974(v=vs.105).aspx. Accessed 6 February 2013.

20. Molen B. Windows Phone 7.5 Mango in-depth preview [online]. Engadget; June 2011.
URL: http://www.engadget.com/2011/06/27/
windows-phone-7-5-mango-in-depth-preview-video. Accessed 6 February 2013.

21. Kadlec T. Implementing Responsive Design: Building sites for an anywhere, everywhere web. New Riders; 2013.

22. Wauters R. eBay: Mobile Saled Grew from $600 Million to $2 Billion in 2010 [online]. TechChrunch; January 2011.
URL: http://techcrunch.com/2011/01/06/ebay-mobile-sales-2010/. Accessed 06 June 2012.

23. Siegler MG. 3 Months to the First Million Users, Just 6 Weeks to the Second Million for Instagram [online]. TechChrunch; February 2011.
URL: http://techcrunch.com/2011/02/14/instagram-2-million/. Accessed 6 June 2012.

24. Ingram M. Mary Meeker: Mobile Internet Will Soon Overtake Fixed Internet [online]. Gigaom; April 2010.
URL: http://gigaom.com/2010/04/12/mary-meeker-mobile-internet-will-soon-overtake-fixed-internet. Accessed 17 May 2013.

25. Bulger D. Smatphone owners: a ready and willing audience [online]. Compete Pulse; March 2012.
URL: http://blog.compete.com/2010/03/12/smartphone-owners-a-ready-and-willing-audience/. Accessed 6 June 2012.

26. Nielsen J., Budiu R. Mobile Usability. New Riders; 2013.

27. Wellman S. Google lays out its mobile user experience strategy [online]. InformationWeek Mobility; April 2007.
URL: http://www.informationweek.com/mobility/business/google-lays-out-its-mobile-user-experien/229216268. Accessed 20 July 2012.

28. Clark J. Tapworthy: Designing iPhone Interfaces for Delight and Usablity [online]. SlideShare; March 2010.
URL: http://www.slideshare.net/joshclark/tapworthy-designing-iphone-interfaces-for-delight-and-usability-3459041. Accessed: 10 February 2013.

29. Wroblewski L. Organizing Mobile [online]. A List Apart; October 2011.
URL: http://alistapart.com/article/organizing-mobile. Accessed 20 March 2013.

30. Online Publishers Association. Opa Study Defines Today's Smartphone User [online]. Online Publishers Association, Press Release. New York, United States; August 2012.
URL: http://onlinepubs.ehclients.com/index.php/opa_news/press_release/opa_study_defines_todays_smartphone_user. Accessed 8 March 2013.

31. Rajapakse C. Techniques for de-fragmenting mobile applications; School of Computing, National University of Singapore; 2008.

32. Native vs. HTML5 Mobile App Development, Appcelerator 2012. Whitepaper.

33. Luo L. Native or Web Application? How best to deliver content and services to your audiences over the mobile phone. White paper. Global Intelligence Alliance, 2010.

34. Mott N. What App Designers Can Learn from a Few Stellar Apps [online]. AppStorm; March 2012.
URL: http://iphone.appstorm.net/general/opinion-general/what-app-designers-can-learn-from-a-few-stellar-apps. Accessed 17 May 2013.

35. Google Play Store [online]. Google; May 2013.
URL: https://play.google.com/store/apps/details?id=com.dropbox.android&hl=en. Accessed 17 May 2013.

36. Windows Phone Store [online]. Microsoft; April 2013
URL: http://www.windowsphone.com/en-us/store/app/disqus/93c35e04-5d4d-42d2-aaaa-3bd1532443b2. Accessed 17 May 2013.

37. Schramm M. Twitter revamps mobile web app [online]. Tuaw; May 2011. URL: http://www.tuaw.com/2011/05/12/twitter-revamps-mobile-web-app. Accessed 17 May 2013.

38. Calimlim A. Google Brings Gmail iOS App's Neat User Interface To Gmail Mobile Web App [online]. AppAdvice; March 2013. URL: http://appadvice.com/appnn/2013/03/ google-brings-gmail-ios-apps-neat-user-interface-to-gmail-mobile-web-app. Accessed 17 May 2013.

39. Kincaid J. YouTube Mobile Goes HTML5, Video Quality Beats Native Apps Hands Down [online]. TechChrunch; July 2010. URL: http://techcrunch.com/2010/07/07/youtube-iphone-mobile-html5. Accessed 17 May 2013.

40. Our Mobile Planet [online]. Google. URL: http://www.thinkwithgoogle.com/mobileplanet. Accessed 17 May 2013.

41. Gemmel M. Apps vs the Web [online]. Matt Gemmell; July 2011. URL: http://mattgemmell.com/2011/07/22/apps-vs-the-web. Accessed 7 March 2013.

42. Online Publishers Association. Survey: Tablet Owners Prefer Browsers to Native Apps [online]. ReadWriteWeb, Titlow; June 2012. URL: http://readwrite.com/2012/06/20/ survey-tablet-owners-prefer-browsers-to-native-apps. Accessed 8 March 2013

43. Hales W. HTML5 and JavaScript Web Apps. O'Reilly Media Inc.; 2012.

44. Raj R., Tolety S., 7-9 Dec. 2012, India Conference (INDICON), 2012 Annual IEEEA study on approaches to build cross-platform mobile applications and criteria to select appropriate approach.

45. Friese P. Cross-Platform Mobile Development [online]. SlideShare; January 2012. URL: http://www.slideshare.net/peterfriese/ cross-platform-mobile-development-11239246. Accessed 7 February 2013.

46. Clancy M, Cremin R, Leonard J. Implementing Your Mobile Strategy whitepaper, dotMobi; 2012.

47. Goasduff L., Pettey C. Gartner Says Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent Growth [online]. Gartner, Press Release. Egham, UK; February 2012. URL: http://www.gartner.com/it/page.jsp?id=1924314. Accessed 20 March 2012.

48. Heitkötter H., Hanschke S., Majchrzak T. Evaluating Cross-Platform Development Approaches for Mobile Applications. Springer Berlin Heidelberg, 2013.

49. Lawson B, Sharp R. Introducing HTML5. Berkley, CA. New Riders; 2011.

50. Rodger R. Beginning Mobile Application Development in the Cloud. Wrox; 2011.

51. Wium Lie, H., C ̧elik, T., Glazman, D., Van Kesteren, A. Media Queries. W3C Recommendation [online]. W3C; June 2012.
URL: http://www.w3.org/TR/2012/REC-css3-mediaqueries-20120619/ Accessed 20 July 2012.

52. Itzkovitch A. Designing For Device Orientation: From Portrait to Landscape [online]. Smashing Magazine; August 2012.
URL: http://uxdesign.smashingmagazine.com/2012/08/10/ designing-device-orientation-portrait-landscape. Accessed 10 December 2012

53. Marcotte E. Responsive Web Design. A Book Apart; 2011.

54. Introducing the new responsive-designed BostonGlobe.com [online]. Filament Group; December 2011.
URL: http://filamentgroup.com/lab/ introduc- ing_the_new_responsive_designed_bostonglobecom. Accessed 17 May 2013.

55. Layon K. The Web Designer's Guide to iOS Applications. New Riders; 2011.

56. Compuware. What Users Want from Mobile. July 2011.

57. Interesting stats [online]. HTTP Archive.
URL: http://www.httparchive.org/interesting.php. Accessed 17 May 2013.

58. Osmani A. Journey Through The JavaScript MVC Jungle [online]. Smashing Magazinze; July 2012.
URL: http://coding.smashingmagazine.com/2012/07/27/ journey-through-the-javascript-mvc-jungle. Accessed 7 December 2012.

59. Koch P. JavaScript library poll results [online]. QuirksMode; October 2012.
URL: http://www.quirksmode.org/blog/archives/2012/10/ javascript_libr_1.html. Accessed 15 October 2012.

60. jQuery Mobile documentation [online]. jQuery Foundation; 2013.
URL: http://view.jquerymobile.com/1.3.0/docs/intro/ Accessed: 10 February 2013.

61. jQTouch documentation [online]. Github.
URL: https://github.com/senchalabs/jQTouch. Accessed: 10 February 2013.

62. Sencha Touch documenation [online]. Sencha Inc.
URL: http://www.sencha.com/products/touch/ Accessed: 10 February 2013.

63. Wroblewski L. Design for Mobile: Native vs. Web Applications [online]. LukeW Ideation + Design; September 2010.
URL: http://www.lukew.com/ff/entry.asp?1193. Accessed 24 March 2012.

64. Kendo UI documentation [online]. Telerik Inc.
URL: http://docs.kendoui.com/getting-started/introduction. Accessed: 10 February 2013.

65. Designing For Device Orientation: From Portrait To Landscape [online]. Smashing Magazine; August 2012.
URL: http://uxdesign.smashingmagazine.com/2012/08/10/designing-device-orientation-portrait-landscape. Accessed: 10 February 2013.

66. iOS Human Interface Guidelines [online]. Apple Inc.; December 2012.
URL: http://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/index.html. Accessed 20 March 2013.

67. Nielsen J., Optimizing a Screen for Mobile [online]. Nielsen Norman Group, Jakob Nielsen's Alertbox; March 2011.
URL: http://www.useit.com/alertbox/mobile-redesign.html Accessed 20 July 2012.

68. Wroblewski L. Responsive Navigation: Optimizing for Touch Across Devices [online]. LukeW Ideation + Design; November 2012.
URL: http://www.lukew.com/ff/entry.asp?1649. Accessed 2 November 2012.

69. Nielsen J. F-Shaped Pattern for Reading Web Content [online]. Nielsen Nor-Norman Group; April 2006.
URL: http://www.nngroup.com/articles/f-shaped-pattern-reading-web-content/ Accessed 12 February 2013.

70. Mozilla. Browser Detection Using the User Agent [online]. Mozilla Developer Network; November 2012.
URL: https://developer.mozilla.org/en-US/docs/Browser_detection_using_the_user_agent. Accessed 10 February 2013.

71. Keight J. It's a Write/Read Mobile Web [online]. Adactio; February 2013.
URL: http://adactio.com/journal/6051/. Accessed: 25 February 2013.

72. Chandler O. What's Going on with Readers Today [online]
URL: http://www.slideshare.net/GoodreadsPresentations/whats-going-on-with-readers-today-16508449. Accessed 10 February 2013.

73. Android. Multi-pane Layouts [online]. Android Developers.
URL: http://developer.android.com/design/patterns/multi-pane-layouts.html Accessed 28 March 2013.

74. Wroblewski L. Multi-Device Layout Patterns [online].
URL: http://www.lukew.com/ff/entry.asp?1514. Accessed 30 March 2013.

75. Tidwell J. Designing Interfaces, second edition. O'Reilly Media Inc.; 2011.

76. Verrecchia J. Build a simple client-side MVC app with RequireJS [online]. November 2011.
URL: http://verekia.com/requirejs/build-simple-client-side-mvc-app-require-js. Accessed 17 May 2013.

77. Technical Note TN2262: Preparing Your Web Content for iPad [online]. Apple, Safari Developer Library; May 2010.
URL: http://developer.apple.com/library/safari/#documentation/ AppleApplica-

tions/Reference/SafariWebContent/UsingtheViewport/ UsingtheViewport.html. Accessed August 20 2012.

78. Rabin J., McCathieNevile C. W3C Mobile Web Best Practices 1.0 [online]. W3C; July 2008.
URL: http://www.w3.org/TR/mobile-bp. Accessed 20 July 2012.

79. Moretti A. Using HTML5 application cache in mobile Web apps [online]. mobiForge; October 2012.
URL: http://mobiforge.com/developing/story/
using-html5-application-cache-mobile-web-apps. Accessed 14 May 2013.

80. Mozilla's Boot 2 Gecko and why it could change the world [online]. Know Your Mobile; March 2012.
URL: http://www.knowyourmobile.com/products/16409/mozillas-boot-2-gecko-and-why-it-could-change-world. Accessed 20 April 2013.

Appendix 1. Mobile Operating Systems Available on the Market

OS Vendors	OSs	Platforms	Browsers
Google	Android	FlashWeb	Android WebKitOpera MobileFirefoxOpera MiniNetFront LifeUC
Samsung	bada	Java MEWeb	DolfinOpera MiniUC
RIM	BlackBerry	Java MEWeb	BB WebKitBolt
RIM	BlackBerry old	Java ME	Opera MiniBoltBB old
Qualcomm	Brew MP	Web	Opera MobileObigoOpera MiniObigo old
Apple	iOS	Web	SafariOpera MiniUC
	LiMo	Web	ObigoNetFrontObigo old
IntelNokia	MeeGo	QtWeb	Opera MobileMicroBQt WebKitFirefox
Lenovo	Ophone		UC
RIM	QNX	Web	BB WebKit
LG	S-class		Phantom
Nokia	S40	Java MEQt	Nokia WebKitQt WebKitOpera MiniOvi
Nokia	Symbian	Java MEQtWeb	Opera MobileNokia WebKitQt WebKitOpera MiniBoltUC
HP	webOS	Web	Palm WebKit
Microsoft	Windows Phone 7		IE7

Copied from http://www.quirksmode.org/mobile/mobilemarket.html

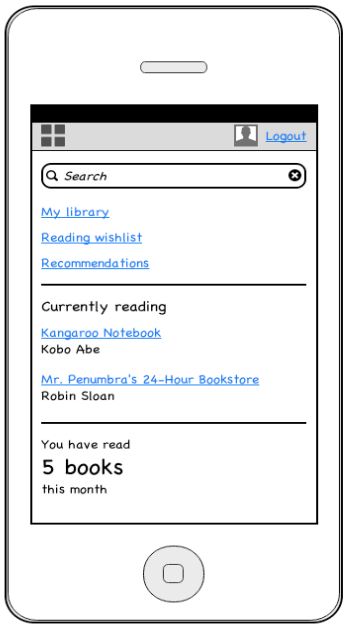
Appendix 2. Top Smartphone Operating Systems, Shipments, and Market Share, Q2 2012

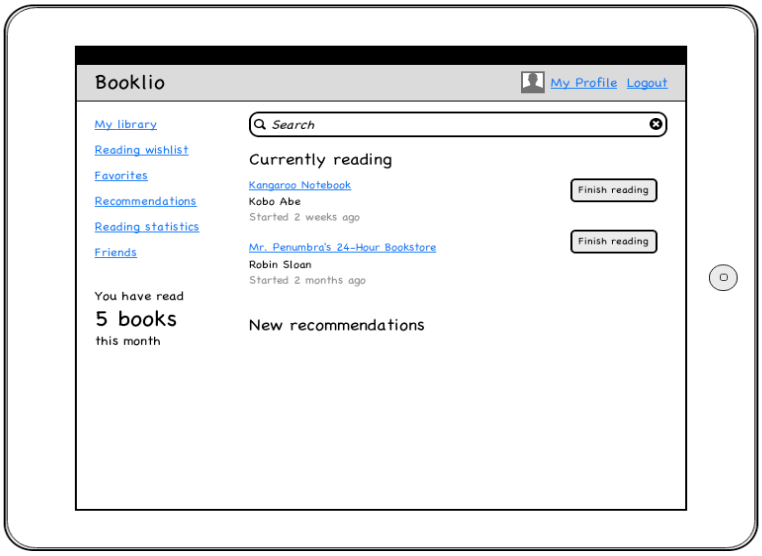
Operating system	Q2 2012 Shipments	Q2 2012 Market Share	Q2 2011 Shipments	Q2 2011 Market Share	Year-over-year change
Android	104.8	68.1%	50.8	46.9%	106.5%
iOS	26.0	16.9%	20.4	18.8%	27.5%
BlackBerry OS	7.4	4.8%	12.5	11.5%	-40.9%
Symbian	6.8	4.4%	18.3	16.9%	-62.9%
Windows Phone 7/ Windows Mobile	5.4	3.5%	2.5	2.3%	115.3%
Linux	3.5	2.3%	3.3	3.0%	6.3%
Others	0.1	0.1%	0.6	0.5%	-80%
Grand total	154.0	100%	108.3	100%	42.2%

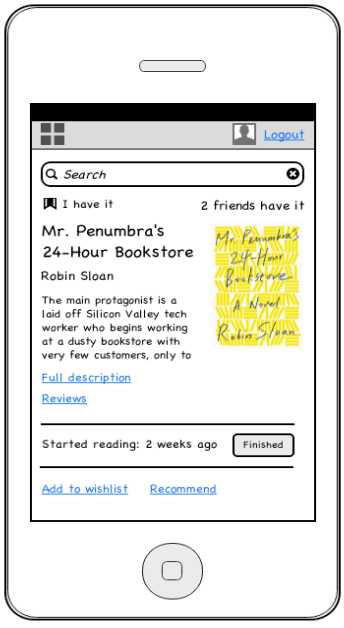
All units are in Millions. Copied from

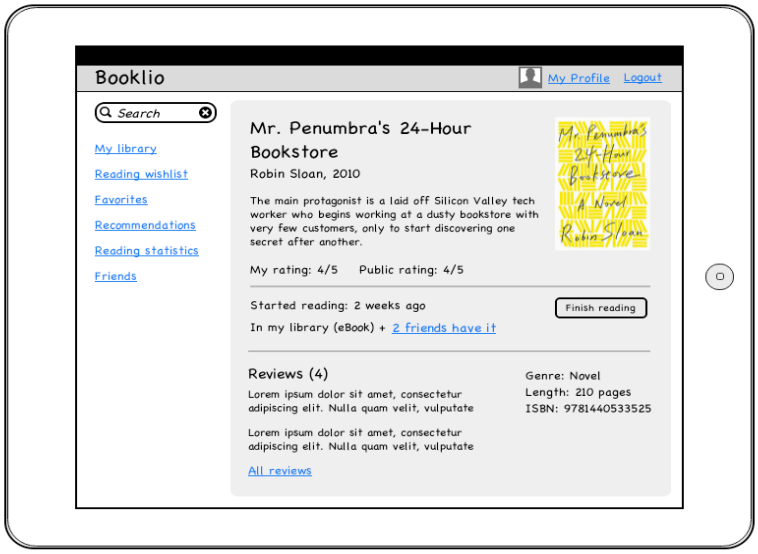
http://www.idc.com/getdoc.jsp?containerId=prUS23638712#.URsW51rWS6F

Appendix 3. Initial Booklio Wireframes for Home and Book Details Wiews

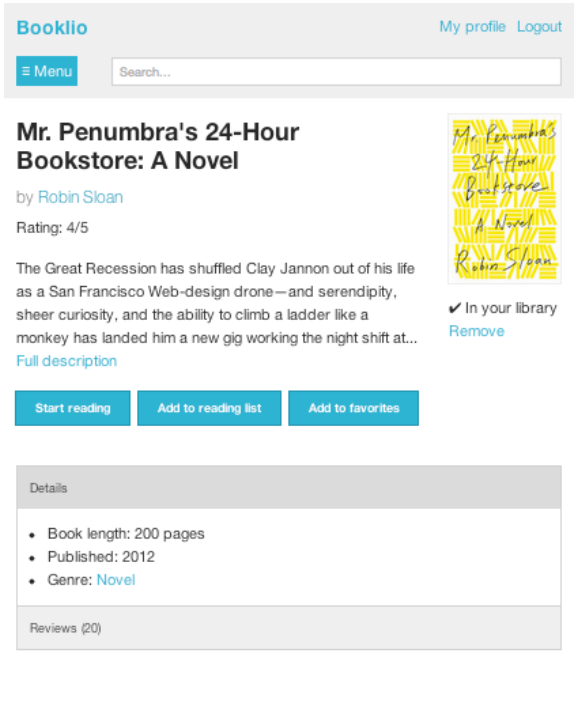


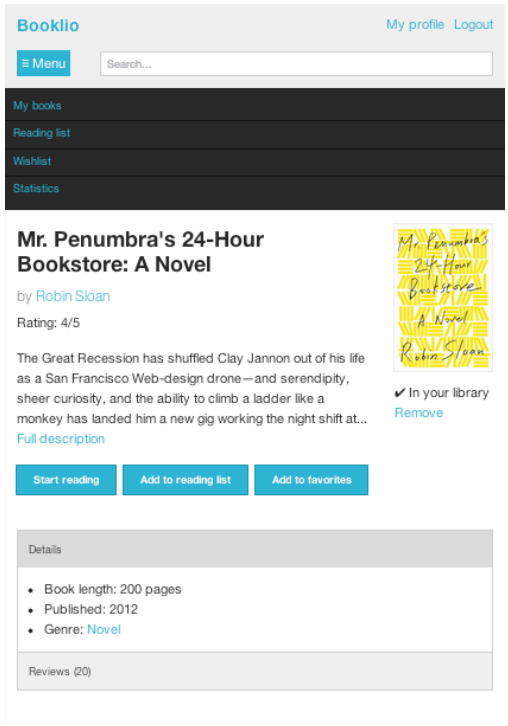


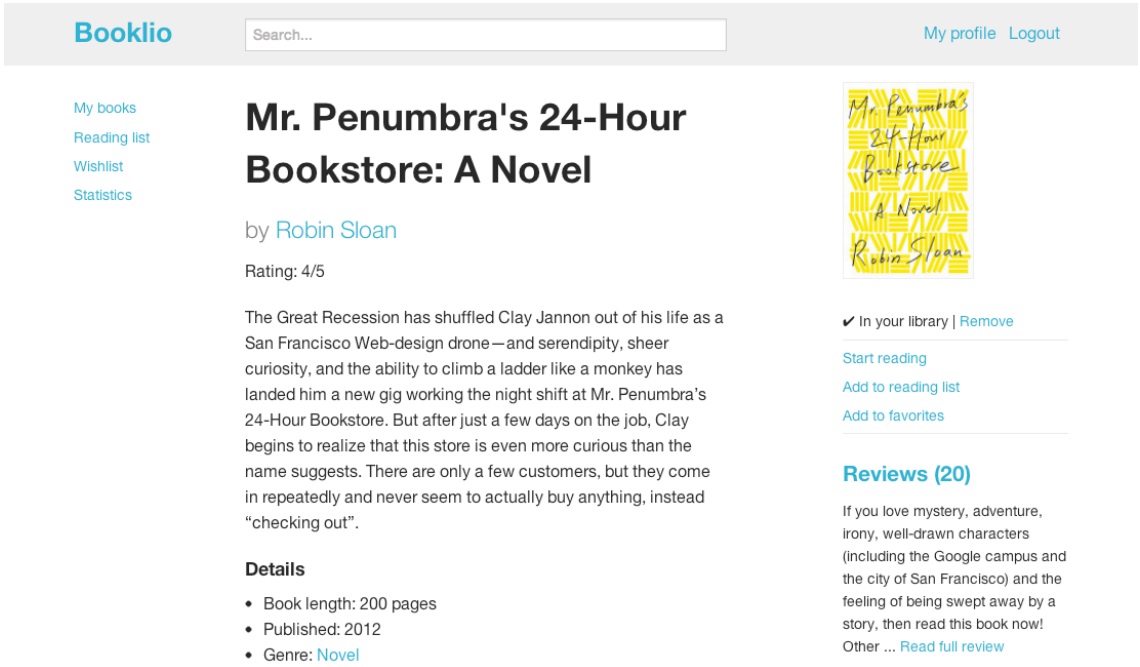




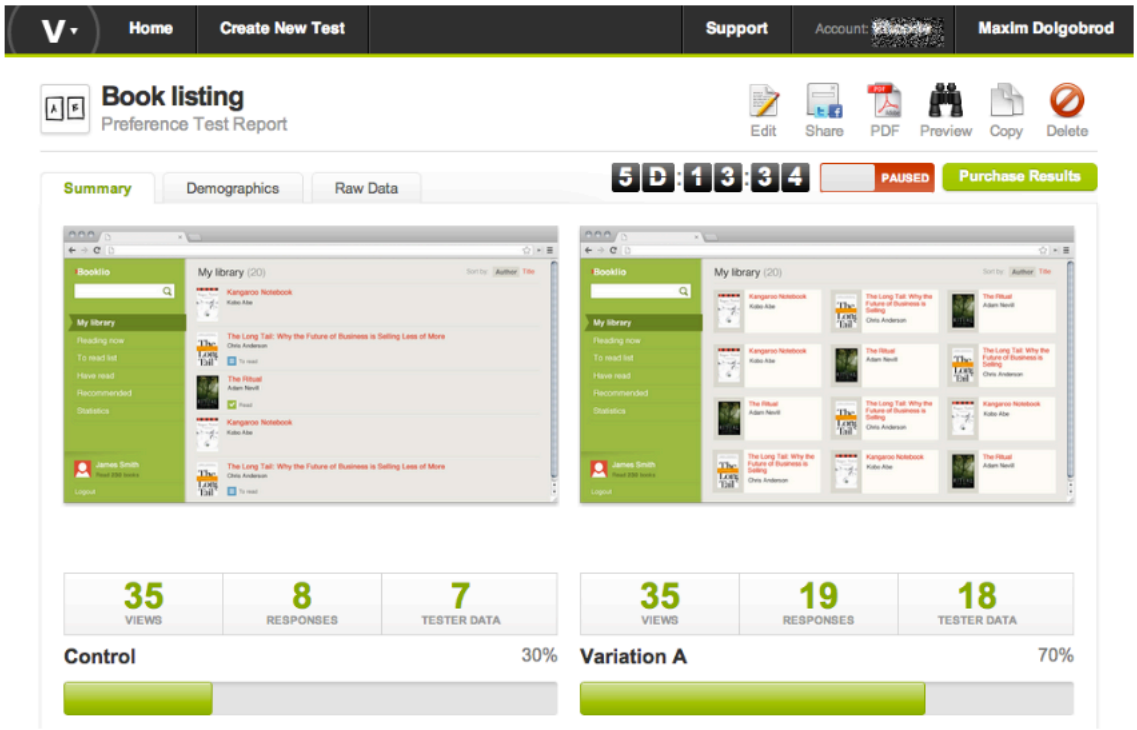
Appendix 4. Booklio HTML Prototype for Books View on Small Screens and Large Screens







Appendix 5. The Booklio User Interface AB Testing using Verify Web Service.



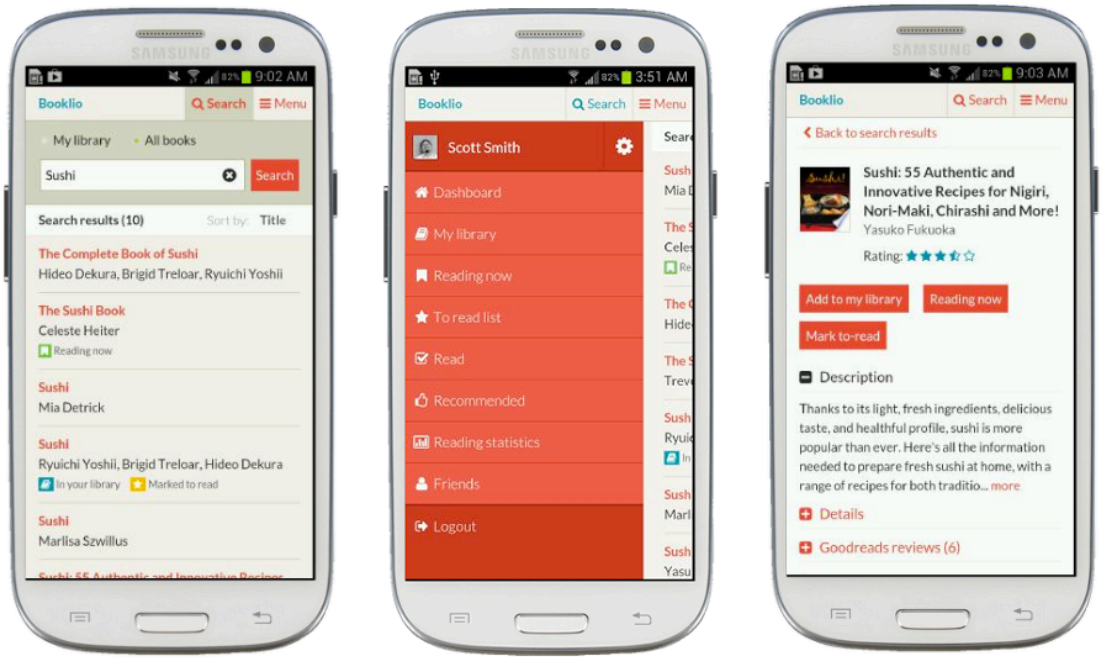
Received comments for each of the versions:

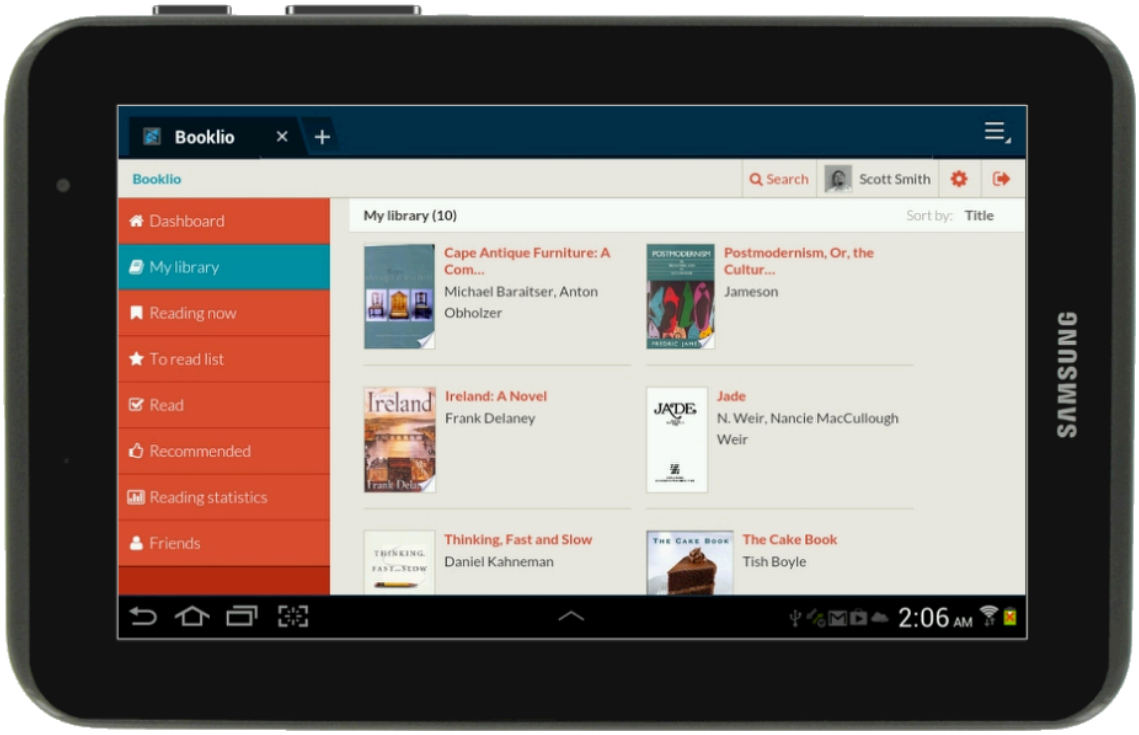
Control or variation A	**Comment**
Control	Simple. Progress icons are nice.
Control	I like that there's more info provided like if you've already read the book or if it's on your 'to read' list. The other page actually had too many books at once. I prefer to be able to scroll through a list rather than have them all crammed on one page.
A	You can see more options at a glance.
Control	It looks easier to see what books are there. But the second design was sabotaged by having repeated images and titles on it. This made it look less interesting and more confusing than it would if it had had unique images and titles for each book.
Control	Is a cleaner layout, and in this way I can read the long titles without effort. Also I feel that for responsive propose will be easy to develop

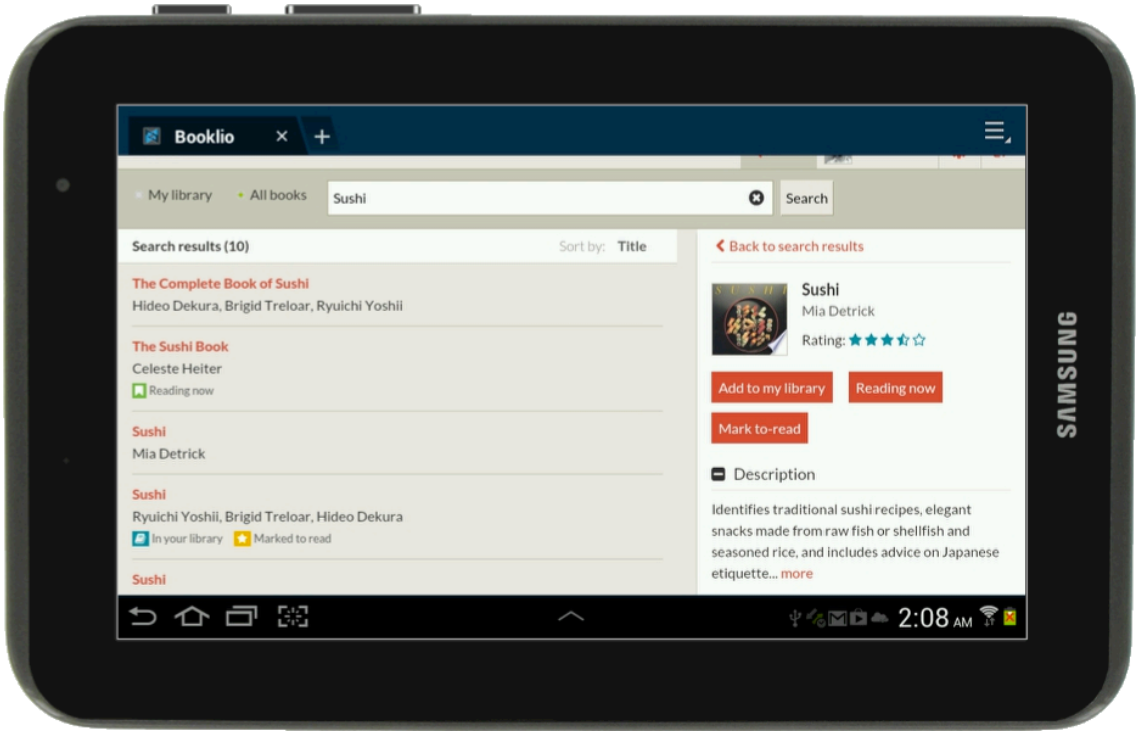
A	Because it makes better use of the page space. (Although I'd like it even more if the listings displayed tags - like the other concept does.) The other one isn't bad at all - but I'd expect a "listings/detail-style view" to have more detail than the "gallery/magazine-style view". I know it has tags, but there's plenty of room for more information. Without more info, it's really just a single column gallery/magazine-style view instead of a true listings/detail-style view.
A	Grid IS legible, and the bookshelf should be as accessible as the real one, which means many covers with the titles. here a customer/reader chooses the cover, then the title, then (eg. with "mouse over" event) he gets lead with the link.
Control	easier to read
A	Because you can see more books than a simple list. It seems to be a waste of space on screen-left. "What about responsive?", you say. A grid can collapse from three columns into two just as easy. Lastly, as a UX designer and typography fan, the list view presents too many letter spaces in a line of text, ~75 letter spaces is the norm.
A	It looks flexible, dynamic, and not boring to see.
Control	cleaner
A	Much prefer grid layout!
A	on a wide screen (>1024) I can see more items without scrolling, just with eye movement. Also items look more divided, with better space allocation
A	I preferred this version as it was more visibly clearly laid out in a grid view compared to the list view which seemed more clustered.
A	I like this version because you can see more of your library at once instead of scrolling repeatedly to see one book at a time.
A	Because this lets me see more books in one viewing, and the information is more evenly distributed throughout the layout.
A	I like that you can see more books at once. However, you need to have a [...] or something at the end of long titles if the whole title doesn't fit in the box. For example the book The Long Tail has a long subtitle and ends with the word "Selling" in this view, but there

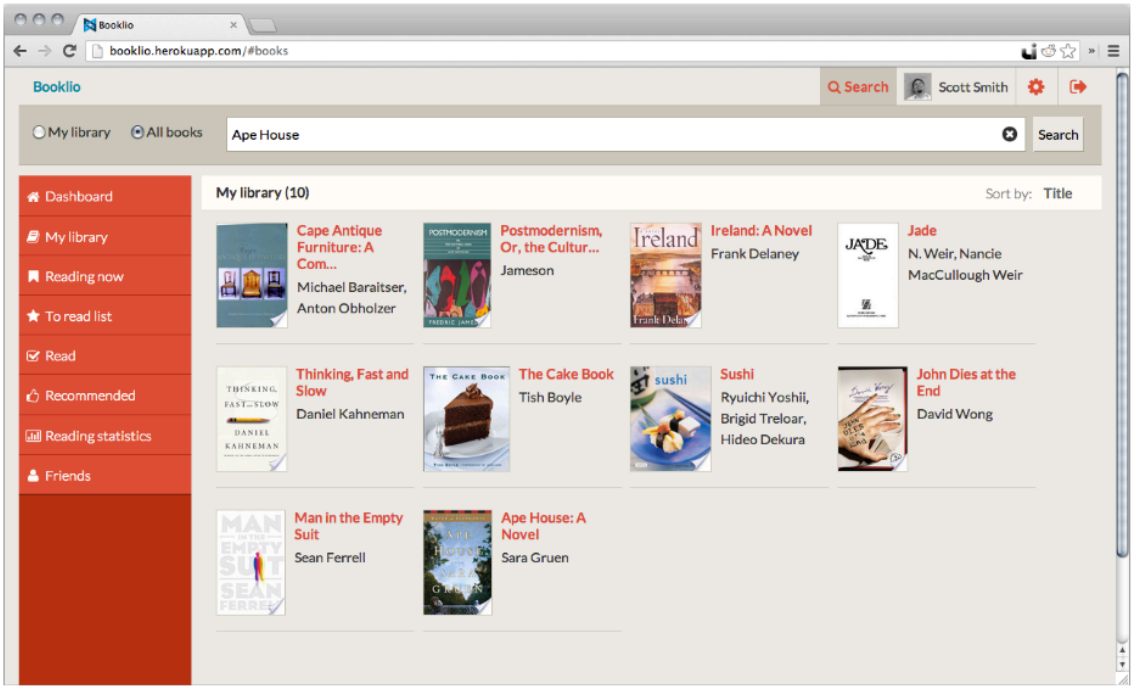
	are actually more words in the title. You need to indicate that the title is longer than what is shown.
A	I like the way the books are laid out so you can find what you want easily.
A	The summary tiles make it quicker to see what's available.
A	The block organisation makes it more viewable and understandable
Control	I can scroll my eyes from top to bottom in a single stroke
A	Less wasted space
A	I like seeing more options at once. It also just seems like a smarter design, better use of space.
A	More items visible on the screen at once, but why without "read" and "to read" marks? Maybe use coners over books' covers?
Control	I'm actually not sure but I think this one because I skim down that single column quicker than I can skim the 3 column even though there's more visible on the 3 column.
A	No comments
A	Better use of space.

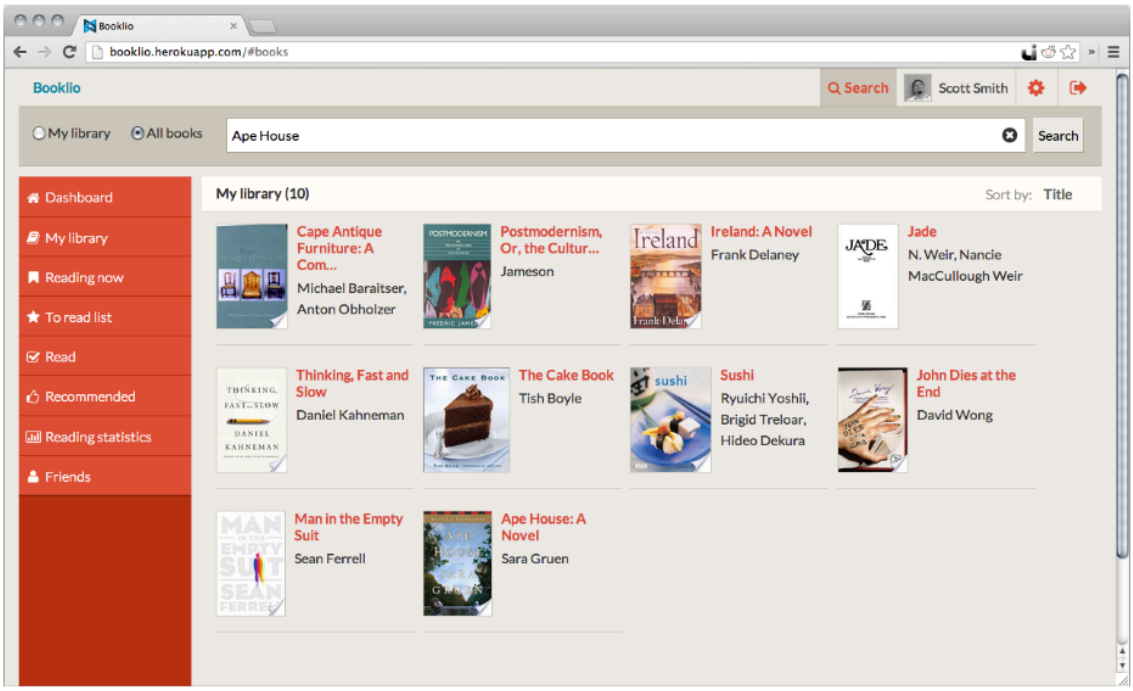
Appendix 6. The Booklio Final Application Prototype on a Smartphone, Tablet and a Desktop Browser.






Booklio
My library
All books
Sushi
Search
Search results (10)
Sort by: Title
The Complete Book of Sushi
Hideo Dekura, Brigid Treloar, Ryuichi Yoshii
The Sushi Book
Celeste Heiter
Reading now
Sushi
Mia Detrick
Sushi
Ryuichi Yoshii, Brigid Treloar, Hideo Dekura
In your library
Marked to read
Sushi
Back to search results
Sushi
Mia Detrick
Rating:
Add to my library
Reading now
Mark to-read
Description
Identifies traditional sushi recipes, elegant snacks made from raw fish or shellfish and seasoned rice, and includes advice on Japanese etiquette... more
2:08 AM
SAMSUNG


Booklio
booklio.herokuapp.com/#books
Booklio
Search
Scott Smith
My library
All books
Ape House
Search
Dashboard
My library
Reading now
To read list
Read
Recommended
Reading statistics
Friends
My library (10)
Sort by: Title
Cape Antique Furniture: A Com...
Michael Baraitser, Anton Obholzer
Postmodernism, Or, the Cultur...
Jameson
Ireland: A Novel
Frank Delaney
Jade
N. Weir, Nancie MacCullough Weir
Thinking, Fast and Slow
Daniel Kahneman
The Cake Book
Tish Boyle
Sushi
Ryuichi Yoshii, Brigid Treloar, Hideo Dekura
John Dies at the End
David Wong
Man in the Empty Suit
Sean Ferrell
Ape House: A Novel
Sara Gruen

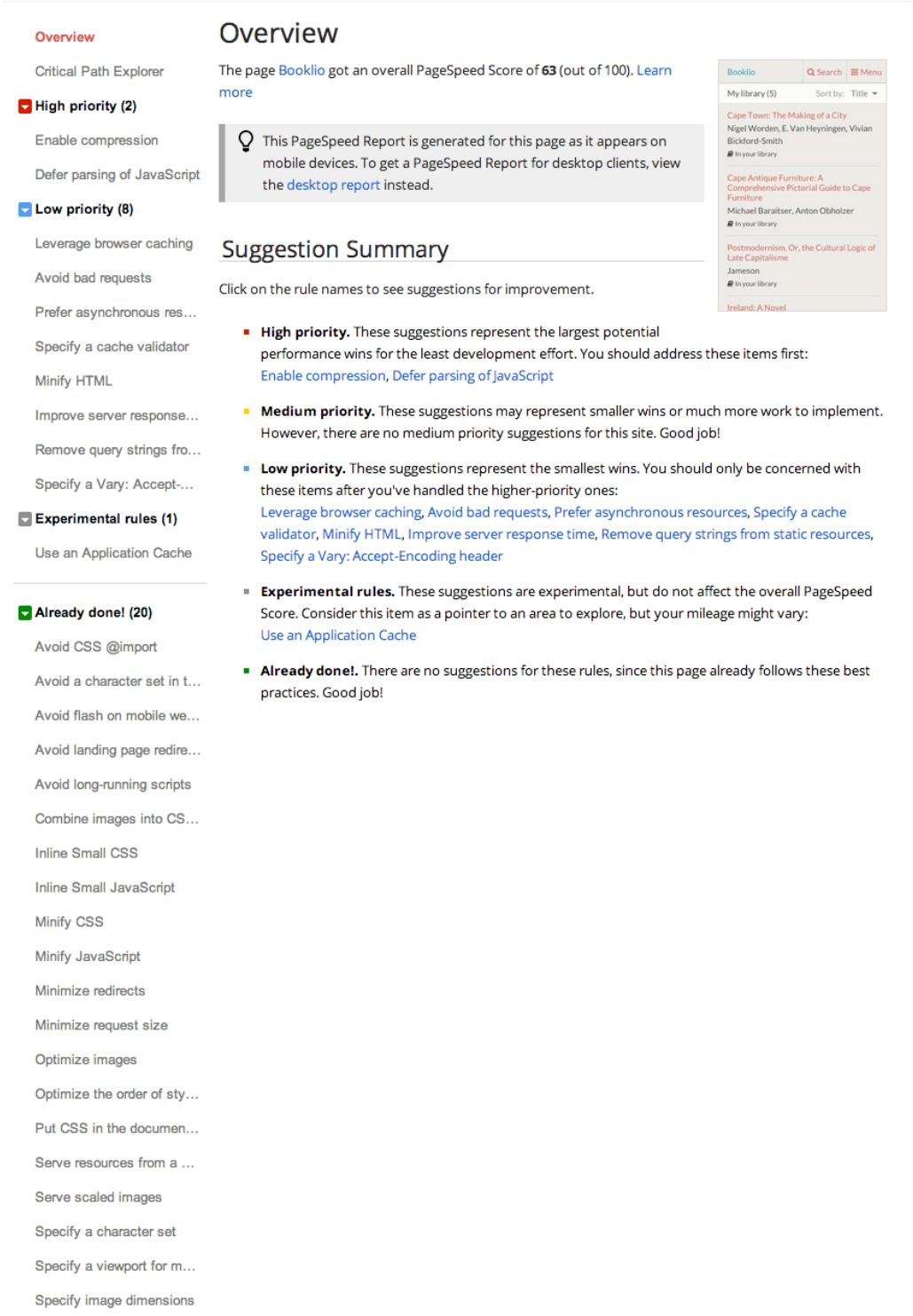

Booklio
booklio.herokuapp.com/#books
Booklio
Search
Scott Smith
My library
All books
Ape House
Search
Dashboard
My library
Reading now
To read list
Read
Recommended
Reading statistics
Friends
My library (10)
Sort by: Title
Cape Antique Furniture: A Com...
Michael Baraitser, Anton Obholzer
Postmodernism, Or, the Cultur...
Jameson
Ireland: A Novel
Frank Delaney
Jade
N. Weir, Nancie MacCullough Weir
Thinking, Fast and Slow
Daniel Kahneman
The Cake Book
Tish Boyle
Sushi
Ryuichi Yoshii, Brigid Treloar, Hideo Dekura
John Dies at the End
David Wong
Man in the Empty Suit
Sean Ferrell
Ape House: A Novel
Sara Gruen

Appendix 7. List of Tested Devices and Found UI Issues

Device Name	Testing via	OS	Browser	UI Issues
Apple iPhone 4S	DeviceAnywhere, Real device	iOS 6.0.1	Safari	No issues
Apple iPhone 5	DeviceAnywhere	iOS 6.0.2	Safari	No issues
Apple iPad	Real device	iOS 4.0.0	Chrome	No issues
Apple iPod	Real device	iOS 4.0.1	Safari	No issues
Motorola Droid2	DeviceAnywhere	Android 2.3.4	Default Android Browser	Minor issue with long titles not fitting properly
Motorola Droid Razr	DeviceAnywhere	Android 4.0.4	Chrome	No issues
Samsung Galaxy Nexus	DeviceAnywhere	Android 4.1.1	Default Android browser	No issues
Samsung Galaxy S II	DeviceAnywhere	Android 2.3.6	Default Android browser	No issues
Samsung Galaxy S III	DeviceAnywhere	Android 4.0.4	Default Android browser	No issues
HTC ONE X	DeviceAnywhere	Android 4.0.0	Default Android browser	No issues
HTC Desire HD	Real device	Android 2.3.5	Default Android browser	No issues
HTC Desire S	Real device	Android 2.3.3	Default Android browser	No issues
LG Thrill 4G	DeviceAnywhere	Android 2.2.2	Default Android browser	Minor issue with long titles not fitting properly
Samsung Galaxy Tab 2	DeviceAnywhere	Android 4.0.3	Default Android browser	No issues
Nokia Lumia 710	DeviceAnywhere	Windows Phone 7.5.0	Internet Explorer 9	Some layout issues; content not loading all the time; icons broken; no animations
MacBook Pro	Real	Mac OS 10.6.8	Chrome 26	No issues
MacBook Pro	Real	Mac OS 10.6.8	Firefox 12	No issues
MacBook Pro	Real	Mac OS 10.6.8	Safari 5.1	No issues
MacBook Pro	Real	Windows 7	Internet Explorer 9	No issue, except broken icons
HP Pavillion	Real	Windows Vista	Internet Explorer 8	Some layout issues; incorrect

				proportions of UI elements; no animations
HP Pavillion	Real	Windows Vista	Chrome	No issues

Appendix 8. Booklio's PageSpeed report for mobile clients



Overview

Critical Path Explorer

High priority (2)

Enable compression

Defer parsing of JavaScript

Low priority (8)

Leverage browser caching

Avoid bad requests

Prefer asynchronous res…

Specify a cache validator

Minify HTML

Improve server response…

Remove query strings fro…

Specify a Vary: Accept-…

Experimental rules (1)

Use an Application Cache

Already done! (20)

Avoid CSS @import

Avoid a character set in t…

Avoid flash on mobile we…

Avoid landing page redire…

Avoid long-running scripts

Combine images into CS…

Inline Small CSS

Inline Small JavaScript

Minify CSS

Minify JavaScript

Minimize redirects

Minimize request size

Optimize images

Optimize the order of sty…

Put CSS in the documen…

Serve resources from a …

Serve scaled images

Specify a character set

Specify a viewport for m…

Specify image dimensions

Overview

The page Booklio got an overall PageSpeed Score of **63** (out of 100). Learn more

This PageSpeed Report is generated for this page as it appears on mobile devices. To get a PageSpeed Report for desktop clients, view the desktop report instead.

Suggestion Summary

Click on the rule names to see suggestions for improvement.

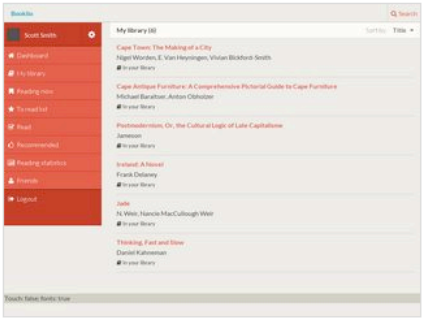
- **High priority.** These suggestions represent the largest potential performance wins for the least development effort. You should address these items first: Enable compression, Defer parsing of JavaScript
- **Medium priority.** These suggestions may represent smaller wins or much more work to implement. However, there are no medium priority suggestions for this site. Good job!
- **Low priority.** These suggestions represent the smallest wins. You should only be concerned with these items after you've handled the higher-priority ones: Leverage browser caching, Avoid bad requests, Prefer asynchronous resources, Specify a cache validator, Minify HTML, Improve server response time, Remove query strings from static resources, Specify a Vary: Accept-Encoding header
- **Experimental rules.** These suggestions are experimental, but do not affect the overall PageSpeed Score. Consider this item as a pointer to an area to explore, but your mileage might vary: Use an Application Cache
- **Already done!.** There are no suggestions for these rules, since this page already follows these best practices. Good job!

Appendix 9. Booklio's PageSpeed report for desktop clients

- Overview
- Critical Path Explorer
- **High priority (1)**
  - Enable compression
- **Low priority (7)**
  - Defer parsing of JavaScript
  - Leverage browser caching
  - Avoid bad requests
  - Prefer asynchronous res…
  - Specify a cache validator
  - Minify HTML
  - Specify a Vary: Accept-…
- **Already done! (20)**
  - Avoid CSS @import
  - Avoid a character set in t…
  - Avoid landing page redire…
  - Avoid long-running scripts
  - Combine images into CS…
  - Improve server response…
  - Inline Small CSS
  - Inline Small JavaScript
  - Minify CSS
  - Minify JavaScript
  - Minimize redirects
  - Minimize request size
  - Optimize images
  - Optimize the order of sty…
  - Put CSS in the documen…
  - Remove query strings fro…
  - Serve resources from a …
  - Serve scaled images
  - Specify a character set
  - Specify image dimensions

Overview

The page Booklio got an overall PageSpeed Score of **67** (out of 100). Learn more



> This PageSpeed Report is generated for this page as it appears in desktop browsers. To get suggestions on how to optimize the performance of this page for mobile devices, generate a mobile report.

Suggestion Summary

Click on the rule names to see suggestions for improvement.

- **High priority.** These suggestions represent the largest potential performance wins for the least development effort. You should address this item first:
  Enable compression
- **Medium priority.** These suggestions may represent smaller wins or much more work to implement. However, there are no medium priority suggestions for this site. Good job!
- **Low priority.** These suggestions represent the smallest wins. You should only be concerned with these items after you've handled the higher-priority ones:
  Defer parsing of JavaScript, Leverage browser caching, Avoid bad requests, Prefer asynchronous resources, Specify a cache validator, Minify HTML, Specify a Vary: Accept-Encoding header
- **Already done!.** There are no suggestions for these rules, since this page already follows these best practices. Good job!