Aalto University
School of Science
!Fixme **Set degree program** Fixme!

Kimmo Puputti

Mobile HTML5

Implementing a Responsive Cross-Platform Application

Master's Thesis
Espoo, !Fixme **Add English date** Fixme!

DRAFT! — Sunday 12th February, 2012 — DRAFT!

Supervisor: Professor Petri Vuorimaa, Aalto University
Instructor: Risto Sarvas D.Sc.(Tech.)

Aalto University
School of Science

Aalto University
School of Science
!Fixme **Set degree program** Fixme!

ABSTRACT OF
MASTER'S THESIS

Author:	Kimmo Puputti		
Title:	Mobile HTML5 Implementing a Responsive Cross-Platform Application		
Date:	!Fixme **Add English date** Fixme!	**Pages:**	vii + 51
Professorship:	Media Technology	**Code:**	T-110
Supervisor:	Professor Petri Vuorimaa		
Instructor:	Risto Sarvas D.Sc.(Tech.)		
!Fixme **Add English abstract** Fixme!			
Keywords:	!Fixme **Add English keywords** Fixme!		
Language:	English		

Aalto-yliopisto
Perustieteiden korkeakoulu

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan tutkinto-ohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Kimmo Puputti		
Työn nimi: !Fixme **Add Finnish title** Fixme! !Fixme **Add Finnish subtitle** Fixme!			
Päiväys:	!Fixme **Add Finnish date** Fixme!	**Sivumäärä:**	vii + 51
Professuuri:	Mediatekniikka	**Koodi:**	T-110
Valvoja:	Professori Petri Vuorimaa		
Ohjaaja:	Tohtori Risto Sarvas		
!Fixme **Add Finnish abstract** Fixme!			
Asiasanat:	!Fixme **Add Finnish keywords** Fixme!		
Kieli:	Englanti		

Acknowledgements

!Fixme **Add acknowledgements** Fixme!
Thank you.

!Fixme **Decide city...** Fixme!, !Fixme **Add English date** Fixme!

Kimmo Puputti

Contents

0.1 Thesis Git repository info

Build time: **Sunday 12th February, 2012 00:18**

Git HEAD:

```
commit 1db4fdc2c512d1ad3989eb868e593905d151eba3
Author: Kimmo Puputti <kpuputti@gmail.com>
Date:   Sat Feb 11 20:29:02 2012 +0200

    Add analytics, timing, and page visibility sections in intro.
```

Repository status:

```
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
# modified:   introduction.tex
# modified:   sources.bib
# modified:   thesis.pdf
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Chapter 1

Introduction

Twenty years after its birth, the Web has become one of the defining technological innovations that knows no geographical, political, or ideological boundaries. The world wide platform built on top of the physical Internet is deeply integrated into our daily lives. This powerful tool that was built on egalitarian principles is now taken for granted, just like old innovations such as electricity. [4]

In parallel with the rapid growth of the Web, mobile phones have evolved from briefcase-sized "portable" telephony devices into modern pocket-sized computers. The mobile revolution has already changed the world as we see it, and more people have access to the Web from a mobile device than from an Internet-connected desktop computer. [13]

The Web is not constrained into (desktop and laptop) computers and mobile phones, though. Tablets, TVs, ebook readers, watches, and even household appliances are connecting to the Internet and have web browsers. For the first time in history, we have a truly ubiquitous digital medium. [13]

Universal accessibility and openness are the keys to being the ubiquitous information platform of the digital age [4]. Now the Web is closer in accomplishing its original principles in equality and universality; anyone can access this vast source of open information from anywhere, with any device. All you need is a web browser that supports the open standards of the Web.

> *The goal of the Web is to serve humanity.*
>
> – Tim Berners-Lee [4]

Being the universal digital medium, mobile devices has some unique characteristics that other mass media lack. Mobile is personal, always-on, always-carried medium with a built-in payment channel. Mobile is in your pocket at the moment you have your creative impulse. [13]

Mobile OS Type	Skill Set Required
Apple iOS	C, Objective C
Google Android	Java (Harmony flavored, Dalvik VM)
RIM BlackBerry	Java (J2ME flavored)
Symbian	C, C++, Python, HTML/CSS/JS
Windows Mobile	.NET
Windows 7 Phone	.NET
HP Palm webOS	HTML/CSS/JS
MeeGo	C, C++, HTML/CSS/JS
Samsung bada	C++

Table 1.1: Required developer skill sets for different mobile platforms according to [8]

These characteristics have made mobile device applications a multibillion-dollar business. Five years after Apple published its game-changing iPhone and the App Store, touch screen mobile phones and tablets from different device manufacturers have spread all over the world. [8, 9, 13]

However, this proliferation of mobile devices and platforms has raised a serious issue for application developers: fragmentation. Not only are there multiple target platforms, but even within the platforms there are different versions with different feature sets, not to mention different devices with varying capabilities. [8]

Table 1 (!Fixme **check table ref number** Fixme!) shows the required developer skill sets for different platforms. As we see, each platform has its own programming language and SDK (?). A lot of knowledge and resources are needed to provide cross-platform applications for these platforms. Making several independent applications with the native tools is also very expensive, and adding features or just maintaining all these different applications becomes costly. [8]

Some developers are forced to make compromises due to resourcing or budgeting, and build their applications only for one platform. This might be fine for independent developers, but a lot of potential customers or users are left out of these walled gardens. Big corporations or public organizations cannot afford leaving out large shares of the mobile market. [4]

!Fixme **add platform market share figure** Fixme!

Being cross-platform is essential in today's mobile market. And if the required skills and resources for the native tools are not present, other options have to be considered. All mobile devices have a web browser, and the Web is becoming the universal application platform. [29, 38]

In the 20 years of its lifetime, the Web has evolved from a simple system

for sharing documents into a massively popular, world wide application and information distribution environment [38]. During the so-called Web 2.0 revolution, the Web grew into a platform for interactive applications with the help of technologies like Ajax (?) [14].

The Web is not without its problems, however. The viral spreading of mobile phones has raised the need for a feature-rich technology stack for building scalable applications that can handle the whole spectrum of devices, screen sizes, and form factors that are used to access the Internet. This is the need that HTML5 (?) with all the related tools and APIs (?) have promised to solve.

Performance is the foundation of a great user experience [8]. By performance we mean the speed of downloading, initializing and using an application as perceived by the user as well as the responsiveness and smoothness of the user interface influencing the overall user experience.

Native tools have been carefully optimized to provide the best possible performance and responsiveness, and web applications are often unfavorably compared to them. In the end, however, the received savings in development time, deployment, cost-efficiency, and cross-platform support can often outweigh the possible compromises. [8, 13]

In this work we look at the performance of HTML5 (?) as a cross-platform application platform for different device form-factors. To study the performance, we built a real world HTML5 application and a JavaScript library and fine-tuned the performance to get the best possible user experience. We then asses these optimizations and the compromises that had to be made.

!Fixme **Add more about background and objectives of this work** Fixme!

1.1 HTML5

HTML5 is a cross-platform and device form-factor agnostic markup language for defining structured documents. It is a backward compatible revision of older HTML standards bringing lots of new functionality, removing unneeded features, and officially documenting some "de facto" standards already supported by some or several web browsers. [31]

In the early 2000s, W3C (?) was developing XHTML (?) and XForms (?) standards to be the future of the Web. Many parts of these standards were backward incompatible and required very strict and error-free authoring. Being frustrated with this vision that was seen as impractical for the real world, a group of web browser vendors and other interested parties had a competing vision of the future of the Web: evolving HTML4 to include

additional features maintaining backward compatibility. W3C members did not agree with this vision, and as a result, the WHAT Working Group was born. [31]

WHATWG (?) is a "loose, unofficial, and open collaboration of Web browser manufacturers and interested parties"[1]. According to a study[2] made by Opera in 2008, more than 95% of web sites do not pass markup validation. Therefore, to maintain backward compatibility and practicality, it is crucial to have a well defined error handling mechanism.

Having the browser vendor and web development community support behind them, after several years the WHATWG work was finally accepted by W3C and a joint effort was started to standardize HTML5. There are still differences in the W3C and WHATWG specifications in what features they include in the main standard and what are separated in other specifications or leaved out, but the main goal is to develop the standards together with browser vendors to get usage feedback while the specifications are being made. This results in many features being available in modern web browsers while the HTML5 and related standards are not yet finished. As a drawback, however, the implementations might change between browser versions, and developers must take extra effort in detecting the supported features. [31]

In this work, we look at HTML5 beyond the main specifications, and take into account also related standards that affect modern web application development. Also, the differences between the W3C and the WHATWG specifications are not separated since they are not clear-cut. This is the practical view that, in our opinion, the web development community has on HTML5.

1.1.1 Semantic Markup

Google did a study[3] in 2005 of a sample of over a billion HTML documents about the popular class names, elements, attributes and related metadata. This analysis had a large impact on which elements and attributes were considered in the upcoming HTML5 standard.

HTML5 defines several new elements and attributes. The objective is to make the markup more semantic for developers and for content processors such as search engines and screen readers.

The specification aims for more semantic structure of HTML by dropping many presentational features. The rationale behind this is explained with the

[1] `http://www.whatwg.org/news/start`
[2] `http://dev.opera.com/articles/view/mama-key-findings/`
[3] `http://code.google.com/webstats/`

following reasons [16]:

- Media-independent markup works for more users and yields better accessibility
- Having style-independent markup separates document structure from its layout and makes maintenance easier
- Separating styling results in smaller document sizes.

Each element in HTML5 is in zero or more content categories that group elements with similar characteristics [16]:

- **Metadata content:** Content that sets the behavior of the document, sets relationships to other documents, or conveys other information of the document.

  Examples: `link`, `meta`, `script`, `title`

- **Flow content:** Most content that are used in the body of a document.

  Examples: `a`, `article`, `audio`, `div`, `header`, `form`, `nav`, `p`

- **Sectioning content:** Content that defines the scope of headings and footers.

  Examples: `article`, `aside`, `nav`, `section`

- **Heading content:** Content that defines a header of a section.

  Examples: `h1`, `h2`, `hgroup`

- **Phrasing content:** Content that holds or marks up the text of the document.

  Examples: `abbr`, `audio`, `canvas`, `img`, `em`

- **Embedded content:** Content that imports another resource or inserts content from another vocabulary into the document.

  Examples: `audio`, `embed`, `iframe`, `img`

- **Interactive content:** Content that is intended for user interaction.

  Examples: `a`, `button`, `menu`, `select`

1.1.2 Extensibility

HTML5 defines the main constructs of a semantic and accessible document. However, some specific use cases require a more precise and context-dependent and fine-grained semantics. Also, web browsers might introduce new features that must conform to the standards. This is why HTML5 is made extensible for adding more semantics or additional features on top of the existing standard.

There are several ways to extend HTML5. The simplest approaches include using the defined general attributes with certain vocabularies. For example, microformats[4] and Schema.org[5] define common elements and class names with certain semantics for defining document metadata.

HTML5 also defines explicit mechanisms for extending the markup structure. Using `data-*=""` and `rel` attributes, `meta` tags, or a generic microdata mechanism, the semantics of the content can be enhanced for automatic reasoning and machine readability. [16]

1.1.3 Media

Multimedia support is crucial for modern applications. HTML5 defines elements and APIs for audio, video, subtitles, and embedded content.

Previously to use these rich content types, developers have had to rely on third party plugins and browser extensions. Not having to rely on plugins and extensions has been one of the main goals of the HTML5 standard for improving the openness and accessibility of web content.

1.1.4 Canvas 2D Context

HTML5 defines the `canvas` element. It is a resolution-dependent bitmap canvas for dynamically rendering graphics. It can be used, for example, for graphs, games, or other visuals. [16]

The Canvas 2D Context specification draft [17] defines a JavaScript API for programmatically drawing on the 2D canvas surface. The API defines functions for drawing shapes, paths, text, gradients, and images on the canvas and other functions for handling the bitmap data.

[4] `http://microformats.org/`
[5] `http://schema.org/`

1.1.5 Form Enhancements

Forms are an essential construction in interactive HTML documents. However, due to their relative simplicity in terms of expressiveness and the lack of proper accessibility features, developers have been forced to build lots of JavaScript solutions to enhance and fix some of these problems.

HTML5 brings several enhancements to forms. New input types for numbers, dates, email addresses, etc. obsolete the need of scripted widgets by using native platform controls. New form attributes like placeholder and autofocus bring easy-to-use accessibility and usability improvements and also diminish the need for scripting. [16]

These additions and enhancements work especially well in mobile context where user input is slow and cumbersome. For example, by having a numeric input field lets the mobile platform open the numeric keyboard by default, which greatly improves the usability of forms. Automatic form validation in the client side also reduces the need for unneeded page refreshes since the browser can show error messages in invalid fields without any JavaScript validation.

1.1.6 Session History Manipulation

HTML was originally designed to be based on documents and hyperlinks between these distinct documents with each of them having a unique URL (?). This hyperlinked structure, however, does not suit well for web applications with dynamic content and interactively changing user interface.

Two of the basic functionalities that users are accustomed to are bookmarking and going back in the session history. Traditionally these have been compromised in dynamic Ajax (?) applications or handled with a lot of extra work.

HTML5 addresses these issues by allowing the developers dynamically manipulate the session history. The history stack can be changed and used for navigation and even the browser address bar can be changed without extra page refreshes. [16]

1.1.7 Offline Web Applications

By design, web sites have always needed a working network connection. Applications, however, should be able to work offline or in unreliable and flaky networks. Especially mobile networks are unreliable (citation needed), which has brought the need for offline support in HTML5.

There are several ways to enable offline support in HTML5 applications. We present these approaches in the following sections.

1.1.7.1 Application Cache

AppCache (?) is a relatively simple way to indicate all resources needed for offline functionality. A manifest file is defined in the HTML document, and within the file there are sections for resources that should always or never be cached and fallback URLs (?) for resources that are not cached but the fetching fails. In addition to the simple manifest file listing offline resources, JavaScript events are defined for cache events.[16]

Example manifest file:

```
CACHE MANIFEST

# Example manifest version 1.

# The resources in this section are cached for offline use.
CACHE:
js/scripts.js
css/styles.css
img/sprite.png
http://example.org/external-image.jpg

# The resources in this section require the user to be online.
NETWORK:
/login

# This section defines resources and their fallback
# URLs if they are inaccessible.
FALLBACK:
/ /offline.html
```

1.1.7.2 Data Storage

Storing data in the client side has traditionally been constrained into using cookies, but HTML5 specifies new options for data persistence.

Two different key/value storages are defined: `localStorage` and `sessionStorage`. The API (?) is same with both of these, but with `sessionStorage`, the data is persisted only for the current browser session. These interfaces are very

simple and easy to use, but are constrained into storing only textual data. [20]

Two more expressive storage APIs (?) have been specified: client side SQL (?) database [20] and the Indexed Database [28]. The client side SQL database defines an asynchronous and transactional SQL database JavaScript API. Although being very expressive, due to the relative complexity compared to simple and scalable key/value storage options, it is yet to be seen if the client side SQL storage will be accepted by the browser vendors and developers.

Indexed Database provides synchronous and asynchronous APIs for storing and searching large amounts of structured data. The transactional API can be used for more complex persistency needs than with the simple key/value storages, and it provides a native JavaScript API that does not involve the complexities of SQL.

1.1.7.3 Detecting Network State

Knowing whether the user is online or offline can affect the user interface or the response to user interactions. HTML5 defines functionality to detect the current network status and events that are fired when the status changes. [16]

Although providing important information, these network status indicators are inherently unreliable [16]. Due to the distributed ad-hoc architecture of the network and possible local or external proxies or middleware, the application can never be sure if the network is connected or not. The only option is just to attempt to make requests and wait for the response or possible failure.

Therefore, applications should be designed to expect the network to work, but to degrade gracefully when the connection is lost or seems to be flaky.

1.1.8 Drag and Drop

Drag and Drop is a common interaction technique where elements can be moved within the user interface from one place to another. Older browsers have had proprietary solutions for this interaction pattern, but HTML5 standardizes the API.

The specification defines the element attributes and DOM (?) events for easily enabling and controlling draggable elements and drop targets. Custom cross-browser JavaScript solutions have enabled this interaction before, but little JavaScript code is needed with the new API. The browser handles the

interaction and the dynamic rendering, reducing the interface lagging and the need for extra processing.

1.1.9 SVG and MathML

While not part of the HTML5 standard, the specification allows for embedding SVG (?) [1] and MathML (?) [7] markup within HTML.

SVG is a markup language for describing two-dimensional vector graphics in XML (?). The markup can be accessed with the DOM (?) API for creating dynamic and interactive functionality. Within an HTML5 document, SVG markup can be embedded within the `svg` element.

MathML is an XML (?) markup language for describing the structure and content of mathematical notation. It can be embedded within an HTML5 document with the `math` element.

Both of these languages reduce the need of custom images, making the content more accessible, dynamic, and enabling dynamic interaction with it. Also, vector graphics can be scaled and fit the available space no matter the screen size, which improves the cross-platform usefulness for different device form factors.

1.2 Other Related Specifications

There are lots of specifications related to HTML5 that are considered to be part of the practical view of all the new Web APIs that are often referred to as 'HTML5'. Some of these originate from the work of the WHATWG (?) and some from the work of W3C (?), some have been part of HTML5 at some point but have been taken out of it into their own separate specifications, and some are just new specifications for the Web that relate to what we can do with HTML, CSS, and JavaScript in modern web applications. In the following sections, we introduce the specifications that are of interest within the topic of this work.

1.2.1 Graphics and Typography

–

1.2.1.1 CSS3

–

1.2.1.2 Media Queries

Older versions of HTML and CSS (?) have already supported targeting stylesheets and rules to certain media types like 'screen', 'print', or 'mobile'. Media queries expand on this technique by adding extra feature queries that can be used to apply styles for certain devices and screen sizes. [44]

Media queries can be used to detect the device screen size and dimensions, orientation, aspect ratio, color depth, etc. [44]. Detecting these media features is especially useful when using the same HTML markup for different device types. For example, a layout of a web application might use more horizontal space when used on a desktop browser, but on a mobile device the fragments of the layout might be stacked vertically to avoid horizontal scrolling. Also background images might be swapped into smaller ones with smaller screens.

1.2.1.3 WebGL and Typed Arrays

WebGL is a low-level 3D rendering API derived from the OpenGL® ES 2.0 [30] and it is designed as a rendering context for the `canvas` element introduced in HTML5. An interactive 3D graphics API in the browser is essential for game development and creates global platform for cross-platform games. WebGL was originally developed by the Khronos Group[6] and later participated by browser vendors and 3D developers. [27]

Being based on the OpenGL ES API, WebGL can run on many different devices, such as desktop computers, mobile phones, tablets, and TVs. The use is not constrained into games only; WebGl can also be used in 3D modeling and design tools, simulations, data visualization, or in interactive art.

Originated from the WebGL specification, typed arrays have been separated into a separate specification [15]. Traditional JavaScript arrays are not typed, but due to performance reasons, WebGL API needed more efficient data structures for 3D graphics. These typed arrays can also be used in non-graphics related contexts, for example, where efficient processing is needed for large amounts of binary data.

1.2.1.4 Web Fonts

Typography is an essential part of design. Traditionally web designers have been constrained into only a few "web safe" fonts that are known to be widely supported between different browsers and platforms.

[6] `http://www.khronos.org/`

CSS3 defines techniques to dynamically load custom fonts and specify their properties [11]. However, different browsers still use different formats and developers might have to provide the custom font files in all the formats that they want to support.

1.2.2 Interaction and Messaging

–

1.2.2.1 Touch Events

User interface events have traditionally relied on the input device being a pointer or a keyboard. Modern mobile devices and tablets, however, usually have a touch screen. Pointer and keyboard events have been mapped into the touch interaction, but proper touch events are needed to support the rich interaction of touch input.

The Touch Events specification [5] defines a set of events for one or more points of contact on a touch surface. The specification defines `touchstart`, `touchmove`, `touchend`, and `touchcancel` events and several new attributes for the event object. These enable developers to add functionality for rich interaction such as swipes and multi-touch gestures.

1.2.2.2 Web Notifications

–

1.2.2.3 Web Messaging

–

1.2.2.4 Fullscreen and Mouse Lock

–

1.2.2.5 Clipboard

–

1.2.3 Files

Local file system access is essential for native applications. JavaScript storage and database APIs can be used for certain structured data for caching and

other purposes, but are cumbersome, for example, for large binary files of arbitrary format. File API specifications [33, 39, 40] define interfaces for creating, reading, writing, and manipulating local files and directories. Error handling and security sandboxing are also specified in the APIs.

Two versions of the file handling APIs are defined: an asynchronous API for normal file handling in the main thread and a synchronous API for file handling in Worker threads (See Section 1.2.7). Text and binary files can be manipulated in memory or as Blob (?) URLs (?) with the DOM (?) API. These capabilities enable web applications to better optimize network transfer and offline support with large files. Combined with the Drag and Drop API (See Section 1.1.8) and Web Workers (See Section 1.2.7), HTML5 forms can be greatly enhanced and optimized with richer interactivity and better performance.

1.2.4 Web Real-time Communication

Support for different multimedia such as audio and video playing is a crucial first step into rich and interactive web applications. However, simply being able to play a video or an audio file within an HTML document is not enough for multimedia rich applications.

Real-time media streaming and playing has been traditionally implemented with Adobe Flash (citation needed)and RTMP (?) (citation needed), but the Web RTC API [3] brings the ability to do native media streaming within HTML documents. Also a direct peer-to-peer streaming communication channel between two user agents is defined.

Combined with the getusermedia API [6], streams can be shown or recorded also from a local media source, such as a web camera. These specifications have lots of security and privacy issues to handle, but they promise very strong multimedia capabilities for web applications.

1.2.5 Web Sockets

Web Sockets API [12, 19] defines a two-way communication protocol for real-time applications between a client, such as web browser, and a remote server. Because HTTP is a stateless protocol, highly interactive applications have introduced many problems when attempting to keep response times and latency low. Real-time applications such as chat clients have been forced to rely on complex workarounds to overcome latency issues.

The Web Socket connection can be open or secure, like HTTP (?) and HTTPS (?). The API uses a single TCP (?) connection that is kept open and allows for traffic in both ways. [12, 19]

The communication protocol specification defines a layer on top of TCP (?) that defines the connection handshaking with HTTP, an "origin"-based security model, addressing and protocol naming mechanism for multiple services on one port and multiple host names on a single IP (?) address, mechanism to overcome TCP packet length limits, and a closing handshake to help deal with proxies and other intermediaries. The intent of Web Sockets is to provide a simple protocol that works well with HTTP and the existing HTTP infrastructure, and that is as close to TCP as possible taking possible security issues into account. [12]

1.2.6 Server-Sent Events

Server-Sent Events specification [18] defines a data stream format `text/event-stream` that can be used to connect an event listener in the client side to listen for events initiated by the server. These streams can be used, for example, for real-time push notifications for data content updates.

The stream data format is very simple, and the API lets the browser handle the message passing. This helps developers to avoid, for example, polling a server for updates, which consumes a lot of computing and networking resources.

1.2.7 Web Workers

The whole IT (?) industry has had a dramatic shift into parallel computing in recent years. Multicore processors have appeared even in mobile devices, and the number of cores is increasing in modern CPUs (?). This parallellization of processing poses challenges and opportunities also for application developers. [2]

Following the trend of the computing industry, together with the proliferation of web technologies, web applications are becoming more capable and processing-intensive. Introducing a traditional threading model and making browser APIs (such as DOM (?)) thread-safe would be an overkill solution and a mismatch to the simplicity and backward compatibility requirements of the Web.

JavaScript is by design single-threaded with an asynchronous event model. The whole user interface also runs in the same thread, and thus long running JavaScript code freezes the whole interface during processing. The traditional approach has been to split the code into small enough pieces, and let the browser handle the scheduling of user interface events and JavaScript processing. This programming model is very hard, and combined with unpredictable browser garbage collection, user interface freezing might be hard

to avoid. [36]

Web Workers are the proposed solution for parallel computing in JavaScript. They introduce a simple interface for sandboxed components with restricted access to browser APIs and an asynchronous communication channel between the worker and the main application. [21]

Web Workers are external JavaScript files initialized from a web site. The worker runs in a separate OS (?)-level thread and does not affect the main UI (?) thread apart from the communication channel. The simple API is easy to work with and gives the long-needed ability of parallel computation in web applications.

1.2.8 Analytics and Timing

Proper analytics is crucial for performance research and optimization. JavaScript timers have traditionally been used for timing and profiling client side interactions, but they can only provide crude analytics of what happens after the browser has started to parse the document and executes the JavaScript timing code.

The Navigation Timing specification [43] defines accurate analytics of events that happens since the user starts to navigate to the target page and until the page is fully loaded. These numbers are much more relevant than timing code supplied by the page itself, since they accurately match the user perceived load time that usually starts already much earlier than what the target page can time, for example, when a link is clicked on another page.

Timing APIs are also defined for resources [24] and general purpose user action profiling [25]. The Performance Timeline specification [23] defines a unifying interface to access these performance metrics.

1.2.9 Page Visibility and Timer Control

Avoiding unneeded work whenever possible is an important performance optimization concept. Modern browsers usually have a tabbed interface with possibly dozens of web pages open at a time. In addition, many interface functionalities use animations and effects to enhance the user experience, but if the page is not visible, these effects have little value and might use the CPU (?) time in vain. Moreover, polling real-time data can use a lot of computing resources even if the data is not visible to the user.

The Page Visibility specification [22] defines an API for JavaScript to know whether the document is currently visible and a `visibilitychange` event to get notified when the document visibility changes. In addition, the Timing control for script-based animations specification [34] and the Efficient

Script Yielding specification [26] define means for requesting the browser to manage the event queue for efficient animation and interface event handling. For example, with the `requestAnimationFrame` function, the developer can periodically request for execution time for smooth animations, and when the page is not visible to the user, the browser can throttle the animation frame frequency for not using CPU (?) power when it might be needed more on another page.

1.2.10 Cross-Origin Resource Sharing

Same-origin policy is an important security restriction in web browsers preventing web applications to obtain data from other origins [41]. However, mashups and applications using data from external data APIs have been very popular in recent years, and techniques have been developed to overcome the restrictions. These techniques have been unsafe and very limited, so a need for cross-domain data sharing has risen.

The Cross-Origin Resource Sharing specification [41] extends the same-origin policy by allowing the HTTP layer to indicate allowed origins for data transfer between separate origins. The specification defines new HTTP headers for indicating allowed origins and for negotiating access control restrictions with preflight requests.

CORS (?) is an important specification for modern web applications, since data is very often distributed to different domains, for example, when using a content delivery network. It also makes mashup development with external data much easier and safer.

Cross-domain resource usage can also be controlled with the means defined in the Content Security Policy specification [37] and The From-Origin Header specification [42]. The specifications define HTTP headers that can be used to restrict certain content types to be allowed only from the given origins.

1.2.11 Device APIs

–

1.2.11.1 Geolocation

Context is one of the main components or a personalized application [13]. Probably the most important aspect affecting the context is the physical location of the user. However, only very crude and error-prone IP (?) based detection techniques have been available for web sites.

The Geolocation API defines a standardized JavaScript API for web applications to query the location of the user. The API is agnostic of the underlying location information sources; the source might usually be the GPS (?) chip on a mobile device or a WiFi (?) network with a known location. The coordinates and the accuracy of them is then provided by the API. [32]

The implementations usually provide some sort of privacy protection by asking the user whether they want to grant the application access to the physical location of the user.

1.2.11.2 Device Orientation

–

1.2.11.3 Gamepad

–

1.2.11.4 User Media

–

1.2.12 Other

–

1.3 Modern Mobile Web Application Architecture

1.3.1 Single-Page applications

–

1.3.1.1 JavaScript MVC Libraries

–

1.3.2 Responsive Design

–

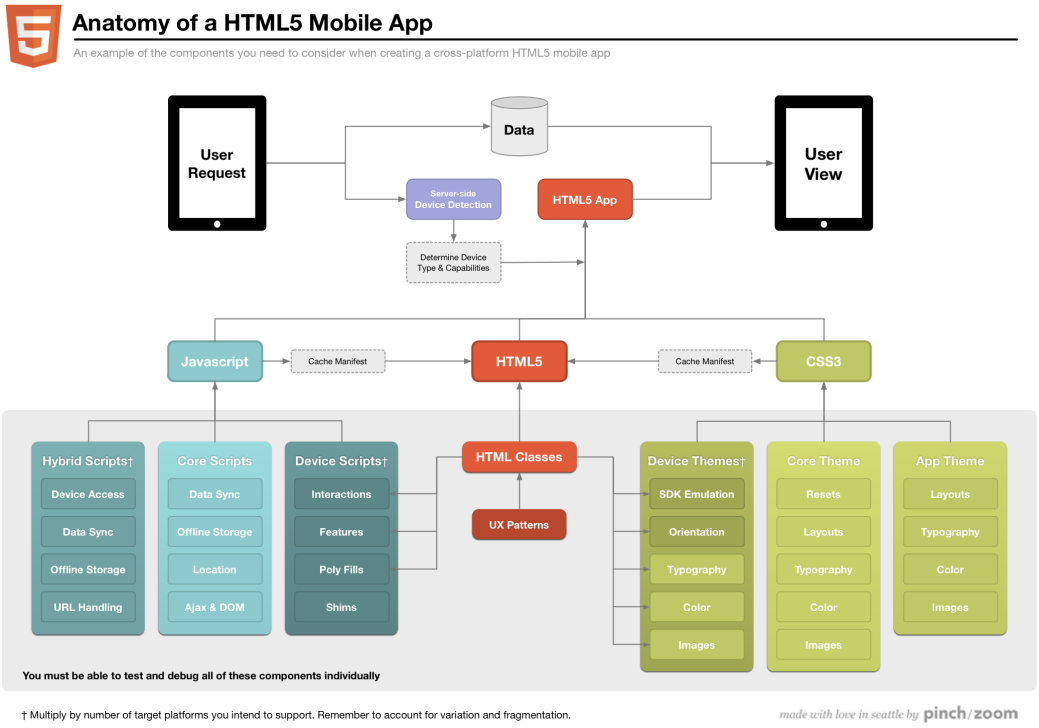

Figure 1.1: HTML5 Mobile Application Anatomy (citation needed)

1.3.3 Progressive Enhancement

–

1.3.4 UI Libraries

–

1.3.4.1 jQuery Mobile

–

1.3.4.2 jQTouch

–

1.3.4.3 Sencha Touch

–

1.3.5 Hybrid Applications

–

1.3.6 Wrapping Web Applications Application Stores

–

1.4 Performance Guidelines

There are several web application performance best practices and related guidelines. According to Souders [35], only 10–20% of the end user response time is spent generating and transferring the HTML document from the web server to the client. Therefore, most of the optimization should be done in the frontend for best improvement opportunities. Below we list the performance guidelines defined by Souders [35, 36].

- **Make Fewer HTTP Requests**

  According to Souders, 80–90% of the end user response time is spent downloading components in a page other than the requested HTML page. Therefore, the simplest way to improve the response time is to reduce the number of HTTP requests needed to get all the required components.

  There are several ways to reducing the number of needed HTTP requests. Combining images into sprites, inlining images, or combining separate JavaScript and CSS files result in fewer components needed to download in a page.

- **Use a Content Delivery Network**

  As web applications are deployed and become accessible worldwide, latency might become an issue for users far from the application's web servers. Geographically distributed servers allow for serving the application as close to the user as possible.

- **Add an Expires Header**

  Avoiding a HTTP request altogether is the best option for reducing the response time when downloading the components in a page. Good caching strategies help browsers to know which resources are valid and for how long until they should be updated.

  The Expires header in HTTP tell the client how long a resource is valid, and especially far future Expires headers reduce the need for downloading an updating the components in a page after the initial download.

- **Gzip Components**

  Compressing HTTP responses is an easy and effective way to reduce the size of the data needed to transfer across the network. Compression is supported widely in web browsers and the impact of reduced response

sizes is huge. Using Gzip, the response size is reduced generally about 70%.

- **Put Stylesheets at the Top**

  Putting the CSS files to the top of the document allows the page to load progressively and the browser show visual feedback to the user as early as possible.

- **Put Scripts at the Bottom**

  Because scripts block parallel downloads, they should be included to the page after all other resources. They also block progressive rendering of all content below them in the HTML document, and should therefore be at the bottom of the document.

- **Avoid CSS Expressions**

  CSS expressions are a way to dynamically set CSS properties in Internet Explorer by evaluating a JavaScript code in a stylesheet. However, despite the obvious upsides, the expressions are evaluated at such a high frequency that they negatively impact the performance.

- **Make JavaScript and CSS External**

  There are performance tradeoffs between making JavaScript and CSS external versus inlining them in the HTML document. In the typical case, however, making them external enables the browser to leverage the HTTP caching semantics and thus reduces the needed network transfer.

- **Reduce DNS Lookups**

  Apart from cached DNS lookups, the browser typically needs 20–120 milliseconds to look up the IP (?) address for a given hostname. The cache lifetime of a lookup depends on the TTL (?) value of the DNS record and having the components of a page distributed across several domains might accumulate into a noticeable response time.

  There is also a trade off between unique hostnames and allowed parallel connections and therefore these settings should be configured based on the application architecture and needs.

- **Minify JavaScript**

  Because JavaScript is an interpreted language that must be sent to the web browser as source code, minifying the code reduces the required

network transfer. Minifiers and obfuscators optimize the size of the source code by stripping extra whitespace and comments as well as renaming variable and function names to shorter ones without changing the interpreted behavior of the code.

- **Avoid Redirects**

  Rerouting any component in a page takes time, and avoiding any kind of redirects improves the response times.

- **Remove Duplicate Scripts**

  Including a resource several times serves no purpose but is actually quite common. Developers should make sure to include resources only once.

- **Configure ETags**

  ETags (?) are a mechanism in HTTP for servers and browsers to validate cached resources. The typical default values set by commonly used web servers might hurt performance, and should thus be configured properly to address the application architecture and needs.

- **Make Ajax Cacheable**

  Highly dynamic web sites have a lot of Ajax (?) [14] functionality, and developers should make sure all the requested URLs for data fetching follow the performance best practices such as having the proper caching in place.

- **Splitting the Initial Payload**

  Nowadays, web sites include a lot of resources and JavaScript functionality, but only a small part of the downloaded components are used in the typical use cases of the application. Splitting the resources into bundles that can be lazily downloaded when first needed reduces the initial payload needed to transfer on application startup.

- **Loading Scripts Without Blocking**

  Most browsers block the downloads of other resources when scripts are being downloaded and executed. There are several ways to circumvent this behavior to allow browsers download scripts in parallel with other resources as well as with other script files.

- **Coupling Asynchronous Scripts**

Related to the previous item, when using parallel downloads with scripts that are dependent on each other, race conditions might occur due to the varying order of download and execution. Therefore, asynchronous scripts dependent on each other should be coupled to preserve the correct order of execution.

- **Positioning Inline Scripts**

  Inline scripts do not introduce a HTTP request, but they can still block parallel downloads of other resources and they might affect also the progressive rendering of the page. With the correct positioning of the scripts, these problems can be handled properly.

- **Writing Efficient JavaScript**

  After networking, the obvious place to optimize the runtime speed of a web application is the JavaScript code.

  Because the whole UI (?) and the JavaScript code run in the same browser thread, there can be only one thing happening at a time. Long running functions block the UI from updating and can result in bad UX (?).

  Splitting the running code into properly sized chunks, appropriately leveraging the asynchronous patterns of JavaScript in the application architecture, understanding the details and slow parts of the DOM API, and using several JavaScript programming best practices can result in big improvements in the perceived application performance. [45]

- **Scaling with Comet**

  For real-time data-driven applications, there are various optimization techniques related to optimizing the constant data transfer between the server and the client. The collection of there various technologies is unofficially called Comet.

- **Going Beyond Gzipping**

  Although Gzipping is widely supported in web browsers, there are cases when it is not supported or when the support is not indicated. Stripping extra content such as unneeded whitespace and comments reduces the payload size for uncompressed responses. There are also ways to detect Gzip support if the client does not directly indicate that.

- **Optimizing Images**

Images typically tend to account for a large portion of the page weight, and since the page weight is highly correlated to the response time, images are a natural target for optimization. There are several ways to optimize images either with lossy or lossless conversions.

- **Sharding Dominant Domains**

  By tuning the amount of unique hostnames used for serving all the resources of an application, parallel downloads can be better leveraged. Also, by using HTTP 1.0 with proper Keep-Alive headers or HTTP 1.1 with proper persistent connections the parallel downloads can be tuned for better performance.

- **Flushing the Document Early**

  Some web application frameworks allow flushing parts of the document to the user before the whole document is generated. This enables progressive rendering and gives faster feedback to the user and thus improves the perceived performance.

- **Using Iframes Sparingly**

  Iframes enable developers to embed a separate HTML document inside another document. They are useful in sandboxing external documents in the same view, but the iframe element is the most expensive DOM element related to the page performance.

- **Simplifying CSS Selectors**

  There are several ways to choose elements in CSS stylesheets to apply the defined properties to. Some selectors are faster than others and some have terrible performance.

Chapter 2

Research Question: HTML5 - Hype versus Realities?

Chapter 3

Methods

3.1 Qt Developer Days 2011 Conference Schedule Application

The Qt Developer Days[1] is a conference for developers using the Qt cross-platform application and UI (?) framework[2]. We created a mobile web application with contextual and personalized session information and daily schedule for the conference.

3.1.1 Requirements

3.1.2 Application Architecture

The conference schedule[3] is a single-page application (citation needed)with a lightweight backend written in Python using the Django Web Framework[4].

The backend provides the static assets (JavaScript, CSS (?), images, etc.) and an API (?) for persisting session feedback to a MySQL[5] relational database. It also generates the HTML5 AppCache (citation needed)offline cache manifest file based on the categorized device type.

The frontend is a JavaScript application written using the Backbone[6] MVC (?) framework. Other used JavaScript libraries include Underscore[7]

[1] http://qt.nokia.com/qtdevdays2011/
[2] http://qt.nokia.com/
[3] http://m.qtdevdays2011.qt.nokia.com/
[4] https://www.djangoproject.com/
[5] http://www.mysql.com/
[6] http://backbonejs.org/
[7] http://underscorejs.org/

for data manipulation, jQuery[8] for DOM (?) API abstraction, Handlebars[9] for templating, and Modernizr[10] for feature detection. The HTML5 Mobile Boilerplate[11] was used as an initial markup structure for the application. The architecture of is depicted in Figure 3.1.

Wireless networks can be unreliable in conference settings, so offline support was also added using several different JavaScript techniques and HTML5 APIs.

The application was designed for touch screens on various platforms and screen sizes. The layout adjusts to the available space and provides rich interactive components. Integration to social networking services was also added as an additional functionality.

!Fixme **add screenshots on different devices (at least phone and tablet** Fixme!

3.2 JSONCache JavaScript Library

JSONCache is a lightweight JavaScript library for fetching JSON (?) data in unreliable networks. The library was designed especially to handle unreliable mobile networks with connection problems and short interruptions. The goal is to avoid networking as long as possible and failing gracefully if the network connections are not stable.

JSONCache provides two main functionalities: data caching and attempting to fetch the data multiple times.

The caching layer uses the client side localStorage (citation needed)cache of HTML5 (?). Data requests can be done using the JSONCache API (?) which always checks the local cache first before opening any network connections. If the data is already in the cache, the cached data is checked for validity and if the data has not been expired, it is returned immediately. If the data is not in the cache or it has been expired, a new network request is made and the received data is cached and returned. The expiration time of a data item can be configured in the library settings.

JSONCache also tries to fetch the data multiple times to handle small interruptions in network connections. !Fixme **add example and explain that it is very common** Fixme!. If a data fetch fails, a new fetch is issued after a timeout (defined in the configuration). On subsequent attempts the

[8] `http://jquery.com/`
[9] `http://handlebarsjs.com/`
[10] `http://www.modernizr.com/`
[11] `http://html5boilerplate.com/mobile`

timeout is increased, and after a defined number of attempts the fetch error is issued.

Figure 3.2 shows an interactive demo of the JSONCache library. The demo[12] simulates the caching and fetching functionality of the library by simulating a unreliable network based on the configuration.

[12] `http://kpuputti.github.com/JSONCache/demo/index.html`

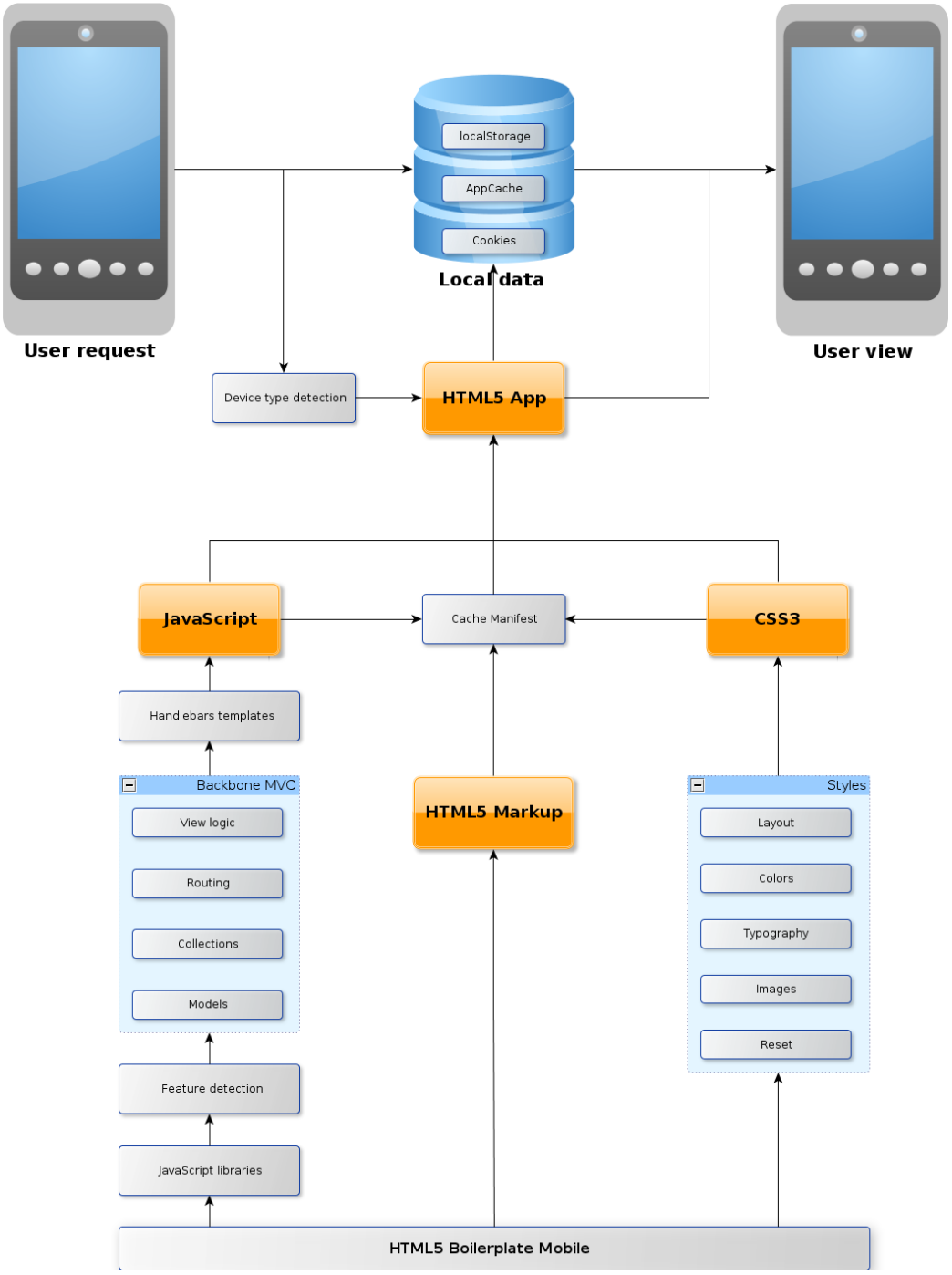

Figure 3.1: Conference schedule application architecture.

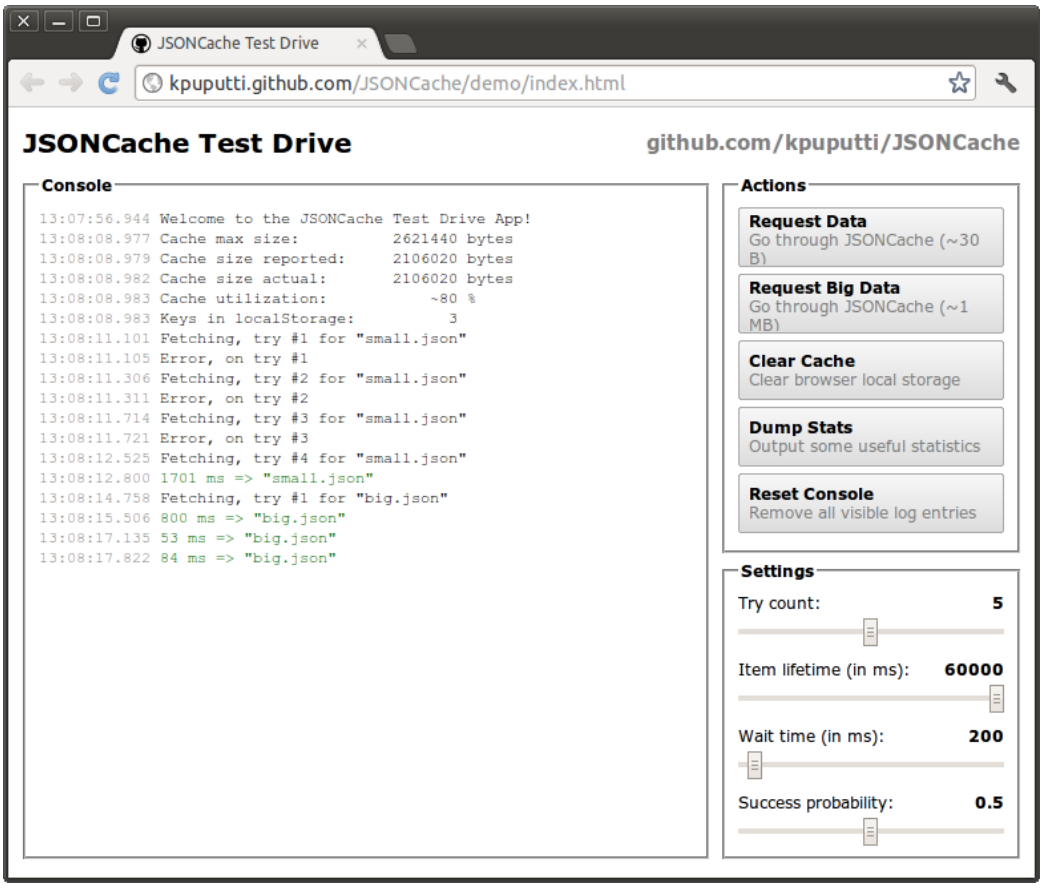

Figure 3.2: Interactive JSONCache demo.

Chapter 4

Results

4.1 Targeting Different Platforms

Despite the web browser being the unified environment for different platforms, there are lots of differences between various devices. The form factors vary from tiny mobile screens to touch screen tablets and desktop monitors and each device and platform has its own feature set. There are also known bugs in the browsers that have to be handled.

Therefore, means to detect the user's device are needed. Here we present two such means: device detection and feature detection. Both of these were used in our conference application.

4.1.1 Device Detection

The User-Agent (UA (?)) HTTP (?) header contains detailed information of the web browser and platform where the request originates. As we can see from Table 4.1.1 (!Fixme **Check table ref number** Fixme!), we can extract platform and browser specific information from the UA header.

In the conference application, device detection was used in the backend to provide a different offline AppCache manifest to different device groups. The detection was also used in defining the assets to be preloaded in the application. The devices were divided into four categories based on the rules defined in Table 4.1.1 (!Fixme **Check table ref number** Fixme!). There were serious limitations in this approach, and compromises had to be made.

First, there is no way to surely know if the device actually is what it reports itself to be. Second, the most important thing to know when generating the screen specific assets in the manifest file would have been the screen size. However, this information is not present in the UA header. We could have listed all the assets for all the devices, but then the list of offline

Device	Platform	User-Agent
Samsung Nexus S	Android 2.3.4	Mozilla/5.0 (Linux; U; Android 2.3.4; en-us; Nexus S Build/GRJ22) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1
Apple iPhone	iOS 3.1.3	Mozilla/5.0 (iPhone; U; CPU iPhone OS 3_1_3 like Mac OS X; de-de) AppleWebKit/528.18 (KHTML, like Gecko) Version/4.0 Mobile/7E18 Safari/528.16
Apple iPad	iOS 5.0	Mozilla/5.0 (iPad; CPU OS 5_0 like Mac OS X) AppleWebKit/534.46 (KHTML, like Gecko) Mobile/9A334
Unknown	Android	Opera/9.80 (Android; Opera Mini/6.5.26571/26.1023; U; de) Presto/2.8.119 Version/10.54

Table 4.1: Example User-Agent strings.

assets would have grown too much and, for example, have large images also for older mobile phones.

Despite the drawbacks, the received advantages of this approach outweighed the possible compromises. The worst that could happen was that the device was wrongly classified and the proper resources were not downloaded for offline use.

Getting platform and browser information from the UA header might look tempting and useful, but it is considered a bad practice to detect a device from it and provide device specific bug fixes or additional features. The header can easily be changed and some browsers or browser plugins even provide preconfigured values for certain browsers or devices for spoofing. Also, the device specific bug fixes might become obsolete with platform updates, and the application might break due to invalid expectations. This is why feature detection is generally the recommended option whenever possible.

4.1.2 Feature Detection

Feature detection is an important concept in Progressive Enhancement design (See Section 1.3.3). A lot of the HTML5 related JavaScript APIs are still unsupported in several platforms, but browser developers are constantly

Rule	Device Type
'iPad' in UA	highres
'iPhone' in UA	iphone
'Android 3' in UA	highres
'mobile' (case insensitive) in UA	mobile
'MIDP' in UA	mobile
'Opera Mobi' in UA	mobile
'Opera Mini' in UA	mobile
otherwise (desktop computer)	highres

Table 4.2: Device type detection rules.

filling the gaps. Therefore, it is important to check whether a certain feature is supported and provide graceful fallback mechanisms for browsers lacking the functionality.

Doing runtime feature detection provides the possibility to give additional functionality to modern browsers and instant support for devices that add the feature support in the lifetime of the application. In the conference application (!Fixme **ref needed?** Fixme!), we used the Modernizr feature detection library (!Fixme **already ref earlier, add bib entry?** Fixme!) to check for HTML5 features.

For example, the user could add sessions to his or her favorites by clicking the star in the agenda or on the session details view (!Fixme **add screenshot?** Fixme!). The favorites were then listed on the home view together with information about the time left for them to begin.

We used HTML5 localStorage for storing the favorites in the user's web browser. By using Modernizr, we detected localStorage support and showed the favorite stars only in browsers that supported the functionality. For all other browsers, the stars were simply hidden and users could not add favorites. We could have also provided a fallback mechanism for persisting the favorites to the backend, but for simplicity and because we targeted mostly modern platforms, this approach was considered as reasonable.

4.2 Targeting Different Screens

Probably the biggest difference in various devices and form factors is the screen size, resolution, and dimensions. Web applications should adjust to the available space and flexible handle screen orientation and window size changes.

First, to target mobile and tablet platforms, the viewport meta informa-

Property	Description	Value
height	Height of the viewport.	pixel value or 'device-height'
width	Width of the viewport.	pixel value or 'device-width'
initial-scale	Initial zoom level.	float value (0.01–10)
minimum-scale	Minimum zoom level.	float value (0.01–10)
maximum-scale	Maximum zoom level.	float value (0.01–10)
user-scalable	Enables/disables zoom.	'yes' or 'no'
target-densitydpi	Visual pixel density.	dpi value, 'device-dpi', 'high-dpi', 'medium-dpi', or 'low-dpi'

Table 4.3: Viewport meta tag configuration for Android.

tion should be indicated in the document. The following tag was used in the conference application:

```
<meta name="viewport" content="width=device-width,
                               initial-scale=1.0">
```

The viewport meta tag was first introduced in Apple's iPhone and afterwards ported to other platforms, such as Android. The possible configuration options and default values might vary between platforms. Values accepted by Android are shown in Table 4.2 (!Fixme **Check table ref number** Fixme!) (citation needed). iOS devices also support these same properties.

If we do not set the viewport configuration tag, the device uses its own default values for the properties. For example, the default value for the width property is 980 pixels in iOS (citation needed), which is clearly defined for web sites targeting desktop browsers. Without setting this value to something smaller and more appropriate in a mobile context, the whole application is very wide and has small and unreadable text in the initial zoom level.

In the viewport configuration we used for the conference application (as defined above), we set the viewport width to 'device_width'. This makes the application width to adjust to the visual pixels of the device screen and works well with screens of different sizes and dimensions. The only other viewport property we set is the initial scaling. This is set to 1.0 to force the browser to render the application without any initial zooming.

In addition to the viewport configuration, we used media queries (citation needed)to use better background images for high resolution screens. We also dynamically set the map view (!Fixme **Add screenshot?** Fixme!) images

based on the screen dimensions so that we could provide smaller images for smaller screens and high resolution images for tablets and other devices with larger screen estate.

4.3 Handling Different Orientations

As shown in the previous section, screen sizes and dimensions vary between devices. In addition to handling different resolutions and dimensions, we must also handle screen orientation changes. The width and height of the touch screens are usually different, and the user can hold the device either in portrait or in landscape mode and in any point switch between these two.

In the conference application, we wanted to have different header and footer background images for different orientations. We also needed to redraw the agenda view when the screen width changes since the items on the schedule needed to be dynamically positioned to the available space.

Mobile browsers fire an 'orientationchange' event whenever the device orientation changes. We listened to this event, inferred the orientation from the screen dimensions, and executed the wanted functionality for the event. We also had to do a fallback for Mobile IE (?) browser to listen to the window resize event because the browser does not support the orientation change event.

4.4 Handling Mobile Networks

One of the biggest problems in mobile web applications is the often slow and unreliable network. Our conference application was designed for a context where the application cannot trust on the networking but should still manage to handle interactions and persist application state. Also being a conference where people come from around the world, the network data transfer cost might be surprisingly high, and thus bandwidth should be saved whenever possible.

4.4.1 Minimizing Data Transfer

The best approach to minimize data that needs to be transferred is to avoid the transfer whenever possible, for example, with proper caching. However, with initial download or with dynamic data, the second best option is to minimize the size of the data needed to be transferred.

First, we made sure the data was minimized and compressed with Gzip. Second, using JSON (?) instead of XML (?) in Ajax (?) requests saves bandwidth and needed effort of the browser to process the data. Third, using the offline manifest ensured that the application assets and data needed to be downloaded only once, and using localStorage we could store the application state locally to the browser avoiding the network completely.

4.4.2 Caching

Caching on different levels of the application stack is one of the most important optimizations that should be done. Caching can be done in the client side using HTML5 APIs, on the HTTP level letting the browser handle it complying to the HTTP caching header semantics, or in various levels of the backend application stack.

In the conference application, we put the most focus on the HTTP caching. Following the performance guidelines specified in Section 1.4, we created unique URLs (?) for all different versions of all static resources (images, CSS, JavaScript, and AppCache manifest files) and set a far future expires header for them. This way we could tell the browser to cache all resources as far as possible and updating the resources was handled by changing the version number in their corresponding URLs.

In addition to the HTTP-level caching, using the AppCache manifest file told the browser to cache all needed resources to a more persistent offline cache, which minimized needed downloads on application startup if the resources were already in the cache.

Client side caching was used in saving the user specific state in the conference application and experimented with the JSONCache library specified in Section 3.2. Using localStorage, we can persist data in the browser and avoid networking if the cached data is still relevant.

JSONCache handles the localStorage caching automatically, with only user configuration needed for setting the data lifetime. Every time the data is rerequested, the local cache is checked first, and networking can be avoided altogether.

4.4.3 Preloading

One way to prevent UI (?) slowness due to flaky networks is to preload resources and data that is expected to be used later on. In the conference application we predownloaded background images and other graphics in the application initialization.

For example, downloading the header and footer background images for both orientations made the device orientation change more responsive because otherwise the browser would have started to download the images after the orientation had already changed. With preloaded images the browser just had to switch the image and render it instantly without any networking.

4.4.4 Offline Support

Using HTML5 AppCache offline manifest file and storing application state to localStorage, we provided full offline compatibility for the conference application. With the offline manifest, we specified the needed resources for all device types as categorized by the rules defined in Section 4.1.1. The offline cache also made subsequent application startups faster since the cache is more persistent than the HTTP cache in browsers.

The offline support was especially critical for the conference application since the conference had indeed very bad wireless network. Without the offline support, users would not have been able to check the session schedule during the conference.

However, the only thing needing the network was the session feedback functionality. The application had a feedback form for all sessions, and the submitted data was persisted in the backend. In offline mode, this functionality was not available. Going further, we could extend the offline support by saving the given feedback, for example, to localStorage and sending it later to the backend server when the network connection is open again.

!Fixme **Add dev headaches: device categories, refreshes in dev, cache updates** Fixme!

4.4.5 Handling Interruptions

Small interruptions are common in mobile networks (citation needed). For example, the user might have a stable network connection, but after walking into an elevator the connection drops for a moment. Then after exiting the elevator the device reconnects to the network. Applications should expect these interruptions and should not fail immediately with brief interruptions in flaky networks.

JSONCache library introduced in Section 3.2 had a functionality to overcome these issues. The library tries to download the requested data multiple times, and fails only when the configured maximum attempt count is reached. With every iteration, a timeout is set for a new request, and the timeout is increased after each failed attempt. This approach works very well, and together with localStorage caching lets data updates circumvent

small network interruptions failing only when the network connection seems to be completely down.

4.5 Animations

Animation and transitions, if not overused, can be a valuable addition to the UX (?) of an application. For example, having a simple sliding animation between different views makes the application more uniform and pleasing to the eye.

There are several ways to animate elements in a web application. The simplest is to use CSS3 (?) animations. However, the performance of the animations is not yet good enough for a cross-platform mobile application. We tried to animate the view changes in our conference application, but even a simple cross-fade did not have good enough performance in all target platforms.

Using progressive enhancement techniques (see Section 1.3.3) we could have provided enhanced experience for the platforms that support animations well, but only iOS devices performed well enough, so for simplicity we did not use any animations.

4.6 Following JavaScript Best Practices

There are lots of best practices and conventions that have been developed by the web developer community. A lot of these tried and tested techniques are outlined in the HTML5 Boilerplate (citation needed)that we used as our base for the conference application. Here we present some techniques and tools that help to improve application performance and reduce bugs.

4.6.1 JSLint

JSLint[1] is a code quality tool for JavaScript. There are several JavaScript features that are suboptimal for performance or code maintainability [10]. JSLint also checks for JavaScript syntax and convention violations, which is valuable because the code will be sent in the source form to be interpreted by the browser.

We had automatic JSLint checking integrated in the Emacs (citation needed)editor that we used for all JavaScript programming, which helped to notice common errors as early as possible and made the code cleaner.

[1] `http://jslint.com/`

4.6.2 Lazy initialization

Postponing work as long as possible is a valuable optimization technique. In lazy initialization we minimize initialization work in application startup to render the initial view fast. We then initialize additional views only when they are requested.

Implementing lazy initialization needs more work than simply doing all initialization work in the application startup, but the received benefits are worth the extra effort. In the conference application we tried to postpone all work to be done as late as possible and doing as little work as possible for faster execution.

4.6.3 Efficient DOM Manipulation

After mobile network issues and data transfers, DOM manipulation is one of the first things to optimize for performance. There are several known performance issues, but the biggest and most common issue is updating a number of elements at once. [45]

The application might, for example, refresh the contents of a list, and with an overly simplistic (but still common) implementation would create the list items and add them to the list container one by one. This causes the browser to reflow the page after each insertion and might add up to UI (?) artifacts and slowness.

One approach to handle updating several elements at once is to use document fragments. With document fragments, several elements can be added to one fragment, which can then be added to the element container. This has no effect on the DOM (?) tree itself, but it requires only one reflow from the browser. One other solution is hiding the container while its contents is modified, and showing after the modification are done. [45]

We used these techniques in the conference application to minimize UI (?) reflows to improve the perceived performance.

4.6.4 Efficient Event Handling

In an interactive web application, there is lots of event listeners and handler functions. For example, a list of dozens or hundreds of items might have one or even several event handlers for each item in the list. This obviously becomes a burden especially in mobile devices with limited processing power and memory.

One way to minimize event listeners is to use event delegation [45]. In event delegation we attach only one event listener to a parent element of the

elements that we want to listen for events. Then in the parent event handler we check the target element of the event and handle the wanted functionality based on the target.

One other optimization for touch screen is to use native touch events instead of traditional mouse events such as click. Mobile browsers typically have a delay or 300 milliseconds after a touchstart event until the the click event is fired (citation needed). This is because the browser waits if the user is doing a double tap instead of a single tap and a delay is needed before a double tap can be excluded. If we bind our event handlers to the touch events instead of click events, we can immediately dismiss this delay altogether and make the UI (?) components a lot more responsive.

We used event delegation and touch events, for example, in the main navigation of the conference application to get the best performance and responsiveness in changing the page views.

4.7 Performance Analysis

We made a quantitative analysis of the conference application performance by using two different tools: YSlow and Page Speed. These tools analyze the web performance practices of a web page and provide optimization guidelines. Many of the rules used in these tools are derived from or based on the guidelines defined by Souders [35, 36] and specified in Section 1.4.

4.7.1 YSlow

YSlow is a website analyzer originally developed by Steve Souders. It checks the website against the rules defined in Section 1.4. We analyzed the conference application performance using YSlow. The results of the analysis are seen in Figure 4.1 and Figure 4.2.

The only sections where an A grade (best grade) was not achieved, were in "Use a Content Delivery Network" and in "Minify JavaScript and CSS". The CDN (?) notice was not seen as important because the application was not designed for world-wide intensive use with lots of users, but rather for a single conference use with some hundreds users.

The minification section notice explained in the details that inline script tag contents should be minified. This was somewhat a limitation in the tool itself, since the inline script tags were the JavaScript templates for the single-page application. The `type` attribute of the tags was `text/x-handlebars-template`, which complies with the generic extending mechanism as specified in the HTML5 specification draft [16]. Thus this was not seen as a problem.

4.7.2 Page Speed

Page Speed (citation needed)is an open-source project by Google for analyzing and optimizing web site performance best practices. We used the Google Chrome browser extension to analyze the conference application against the performance rules defined in Page Speed. The results are pictured in Figure 4.3.

We were very happy with the Page Speed score of 92 out of 100. A lot of the performance rules analyzed by Page Speed are similar to the guidelines listed in Section 1.4, but there are also additional rules.

The only real problem in the score was the 'Optimize Images' rule. We had not optimized the images used in the application, but instead used the images provided by the designers. Going further, we could have saved a lot or bandwidth by optimizing the images with tools such as Pngcrush[2].

Of the other notes in the results, 'Defer parsing of JavaScript' could have been avoided by adding a 'defer' attribute to all the script tags in the document. The reason for this rule is that scripts block page rendering as defined in Section 1.4. However, since we followed the guideline 'Put Scripts at the Bottom', this rendering issue is avoided. The only script in the document head was Modernizr, which must be included before the page is parsed because it creates the essential HTML5 tag support for older browsers and must do so before the tags are parsed.

The 'Minify JavaScript' note was probably due to the Handlebars templating library not being minified. All the JavaScript libraries were included in their minified form, but Handlebars library was only available unminified. We also did not want to minify it ourselves to avoid breaking any functionality. All other JavaScript files were minified and concatenated to avoid extra HTTP requests.

The 'Minify CSS' note was not seen as important since CSS compression does not yield big improvements and because the CSS files were already Gzipped delivered. The 'Remove query strings from static resources' note means that query parameters like '?123' should be removed from the end of the URLs (?) because they might not be cached in some proxies. We did not change this because the query strings in the static assets were an essential part of our caching strategy.

4.8 Conclusions

[2] `http://pmt.sourceforge.net/pngcrush/`

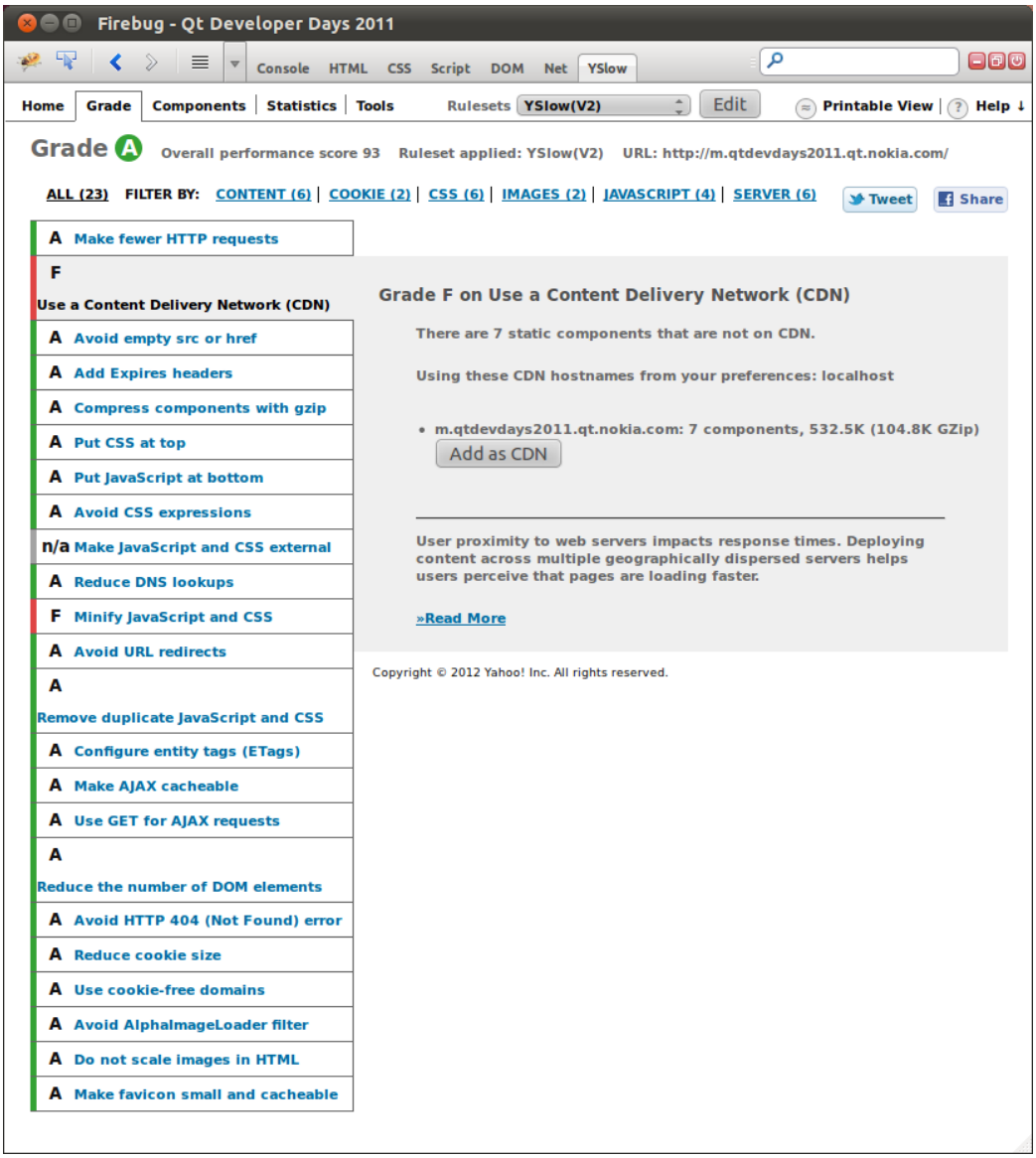

Figure 4.1: YSlow results grade.

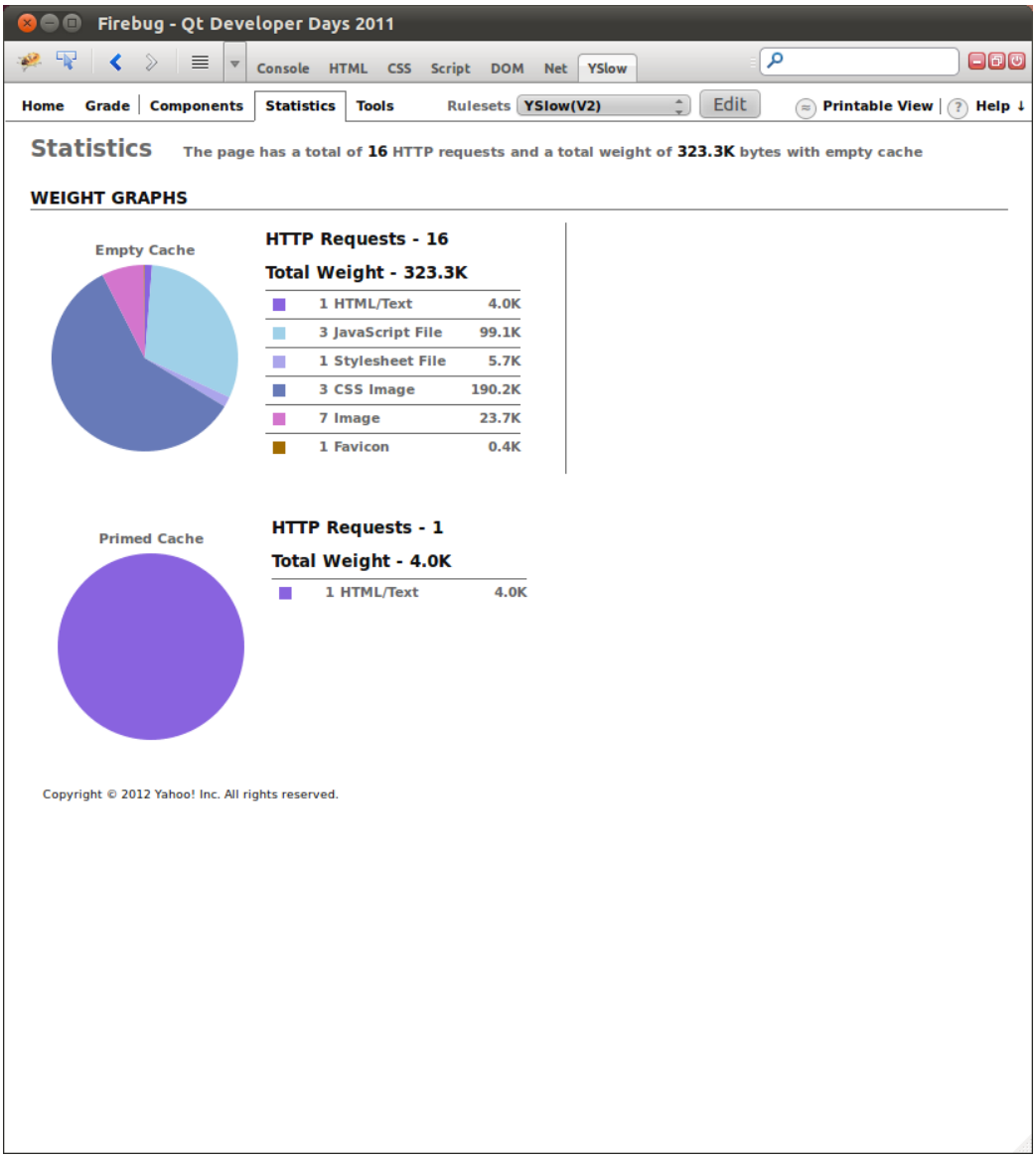

Figure 4.2: YSlow results statistics.

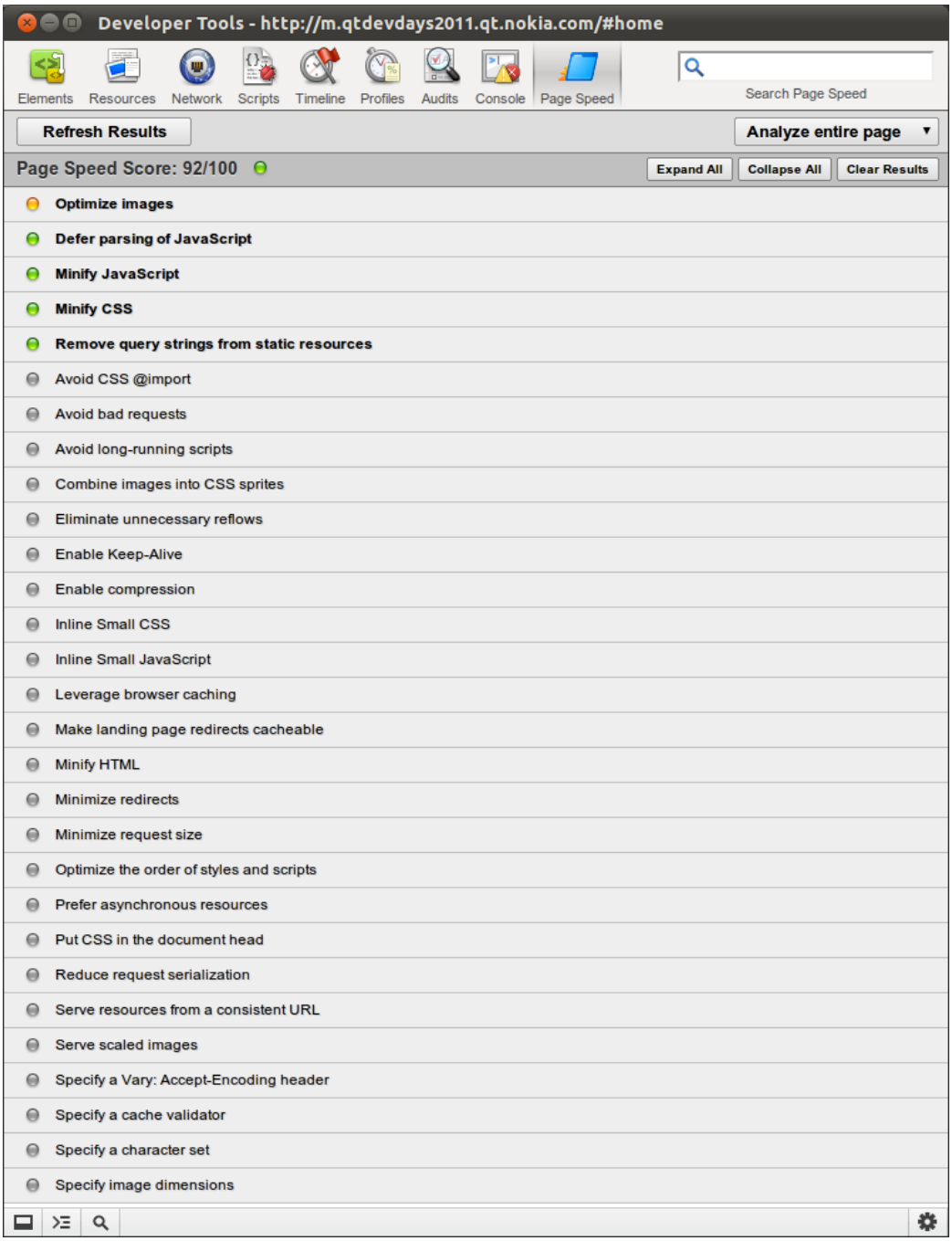

Figure 4.3: Page Speed results for the conference application.

Chapter 5

Discussion: Bright Future Ahead for HTML5

- validate results
- discuss about further work or possible improvements

> *Context is what makes mobile such a powerful and exiting medium. It takes the foundational concepts of western civilization and evolves them to create a new means of understanding, or what ancient Creeks called inventio, meaning both invention and discovery.*
>
> – Brian Fling [13]

Bibliography

[1] Andersson, O., et al. Scalable Vector Graphics (SVG) Tiny 1.2 Specification. W3C Recommendation, W3C, Dec 2008. Available at: `http://www.w3.org/TR/SVGTiny12/`. Accessed 6-Febuary-2012.

[2] Asanovic, K., Bodik, R., Demmel, J., Keaveny, T., Keutzer, K., Kubiatowicz, J., Morgan, N., Patterson, D., Sen, K., Wawrzynek, J., et al. A View of the Parallel Computing Landscape. *Communications of the ACM 52*, 10 (2009), 56–67.

[3] Bergkvist, A., Burnett, D. C., Jennings, C., and Narayanan, A. WebRTC 1.0: Real-time Communication Between Browsers. W3C Editor's Draft, W3C, Jan 2012. Available at: `http://dev.w3.org/2011/webrtc/editor/webrtc.html`. Accessed 11-Febuary-2012.

[4] Berners-Lee, T. Long Live the Web. *Scientific American 303*, 6 (2010), 80–85.

[5] Brubeck, M., Moon, S., and Schepers, D. Touch Events version 1. W3C Candidate Recommendation, W3C, Dec 2011. Available at: `http://www.w3.org/TR/touch-events/`. Accessed 7-Febuary-2012.

[6] Burnett, D. C., and Narayanan, A. getusermedia: Getting access to local devices that can generate multimedia streams. W3C Editor's Draft, W3C, Dec 2011. Available at: `http://dev.w3.org/2011/webrtc/editor/getusermedia.html`. Accessed 11-Febuary-2012.

[7] Carlisle, D., Ion, P., and R., M. Mathematical Markup Language (MathML) Version 3.0. W3C Recommendation, W3C, Oct 2010. Available at: `http://www.w3.org/TR/MathML/`. Accessed 6-February-2012.

[8] Charland, A., and Leroux, B. Mobile Application Development: Web vs. Native. *Communications of the ACM 54*, 5 (2011), 49–53.

[9] Cortimiglia, M., Ghezzi, A., and Renga, F. Mobile Applications and Their Delivery Platforms. *IT Professional 13*, 5 (2011), 51–56.

[10] Crockford, D. *JavaScript: The Good Parts*. O'Reilly Media / Yahoo Press, 2008.

[11] Daggett, J. CSS Fonts Module Level 3. W3C Editor's Draft, W3C, Oct 2011. Available at: `http://dev.w3.org/csswg/css3-fonts/`. Accessed 7-Febuary-2012.

[12] Fette, I. RFC 6455: The WebSocket protocol. *Status: Internet Draft*. Available at: `http://tools.ietf.org/html/draft-ietf-hybi-thewebsocketprotocol-17`. Accessed 8-February-2012.

[13] Fling, B. *Mobile Design and Development: Practical Techniques for Creating Mobile Sites and Web Apps*. O'Reilly Media, Inc., 2009.

[14] Garrett, J. J. Ajax: A New Approach to Web Applications. *Adaptive path 18* (2005). Available at: `http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications`. Accessed 5-January-2012.

[15] Herman, D., and Russell, K. Typed Array Specification. Editor's Draft, Dec 2011. Available at: `https://www.khronos.org/registry/typedarray/specs/latest/`. Accessed 8-Febuary-2012.

[16] Hickson, I. HTML 5. W3C Editor's Draft, W3C, Jan 2012. Available at: `http://dev.w3.org/html5/spec/Overview.html`. Accessed 31-January-2012.

[17] Hickson, I. HTML Canvas 2D Context. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dev.w3.org/html5/2dcontext/`. Accessed 2-February-2012.

[18] Hickson, I. Server-Sent Events. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dev.w3.org/html5/eventsource/`. Accessed 11-Febuary-2012.

[19] Hickson, I. The WebSocket API. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dev.w3.org/html5/websockets/`. Accessed 8-Febuary-2012.

[20] Hickson, I. Web Storage. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dev.w3.org/html5/webstorage/`. Accessed 6-February-2012.

[21] Hickson, I. Web Workers. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dev.w3.org/html5/workers/`. Accessed 9-Febuary-2012.

[22] Mann, J., and Jain, A. Page Visibility. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dvcs.w3.org/hg/webperf/raw-file/tip/specs/PageVisibility/Overview.html`. Accessed 11-Febuary-2012.

[23] Mann, J., and Wang, Z. Performance Timeline. W3C Editor's Draft, W3C, Jan 2012. Available at: `http://dvcs.w3.org/hg/webperf/raw-file/tip/specs/PerformanceTimeline/Overview.html`. Accessed 11-Febuary-2012.

[24] Mann, J., Wang, Z., and Quach, A. Resource Timing. W3C Editor's Draft, W3C, Jan 2012. Available at: `http://www.w3c-test.org/webperf/specs/ResourceTiming/`. Accessed 11-Febuary-2012.

[25] Mann, J., Wang, Z., and Quach, A. User Timing. W3C Editor's Draft, W3C, Jan 2012. Available at: `http://dvcs.w3.org/hg/webperf/raw-file/tip/specs/UserTiming/Overview.html`. Accessed 11-Febuary-2012.

[26] Mann, J., and Weber, J. Efficient Script Yielding. W3C Editor's Draft, W3C, Jul 2011. Available at: `http://dvcs.w3.org/hg/webperf/raw-file/tip/specs/setImmediate/Overview.html`. Accessed 11-Febuary-2012.

[27] Marrin, C. WebGL Specification. Editor's Draft, Jan 2012. Available at: `https://www.khronos.org/registry/webgl/specs/latest/`. Accessed 8-Febuary-2012.

[28] Mehta, N., Sicking, J., Graff, E., Popescu, A., and Orlow, J. Indexed Database API. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dvcs.w3.org/hg/IndexedDB/raw-file/tip/Overview.html`. Accessed 6-February-2012.

[29] Mikkonen, T., and Taivalsaari, A. Apps vs. Open Web: The Battle of the Decade. In *2nd Annual Workshop on Software Engineering for Mobile Application Development* (2011).

[30] Munshi, A., and Leech, J. OpenGL® ES Common Profile Specification Version 2.0.25. Common Profile Specification, Nov 2010. Available at: `http://www.khronos.org/registry/gles/specs/2.0/es_full_spec_2.0.25.pdf`. Accessed 8-Febuary-2012.

[31] Pilgrim, M. *HTML5: Up And Running*. O'Reilly Media / Google Press, 2010.

[32] Popescu, A. Geolocation API Specification. W3C Candidate Recommendation, W3C, Sep 2010. Available at: `http://www.w3.org/TR/geolocation-API/`. Accessed 6-Febuary-2012.

[33] Ranganathan, A., and Sicking, J. File API. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dev.w3.org/2006/webapi/FileAPI/`. Accessed 10-Febuary-2012.

[34] Robinson, J., and McCormack, C. Timing control for script-based animations. W3C Editor's Draft, W3C, Jan 2012. Available at: `http://dvcs.w3.org/hg/webperf/raw-file/tip/specs/RequestAnimationFrame/Overview.html`. Accessed 11-Febuary-2012.

[35] Souders, S. *High Performance Web Sites*. O'Reilly Media, 2007.

[36] Souders, S. *Even Faster Web Sites*. O'Reilly Media, 2009.

[37] Sterne, B., and Barth, A. Content Security Policy. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dvcs.w3.org/hg/content-security-policy/raw-file/tip/csp-specification.dev.html`. Accessed 12-Febuary-2012.

[38] Taivalsaari, A., and Mikkonen, T. The Web as an Application Platform: The Saga Continues. In *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on* (2011), IEEE, pp. 170–174.

[39] Uhrhane, E. File API: Directories and System. W3C Editor's Draft, W3C, May 2011. Available at: `http://dev.w3.org/2009/dap/file-system/file-dir-sys.html`. Accessed 10-Febuary-2012.

[40] Uhrhane, E. File API: Writer. W3C Editor's Draft, W3C, May 2011. Available at: `http://dev.w3.org/2009/dap/file-system/file-writer.html`. Accessed 10-Febuary-2012.

[41] van Kesteren, A. Cross-Origin Resource Sharing. W3C Editor's Draft, W3C, Dec 2011. Available at: `http://dvcs.w3.org/hg/cors/raw-file/tip/Overview.html`. Accessed 11-Febuary-2012.

[42] van Kesteren, A. The From-Origin Header. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dvcs.w3.org/hg/from-origin/raw-file/tip/Overview.html`. Accessed 12-Febuary-2012.

[43] Wang, Z. Navigation Timing. W3C Editor's Draft, W3C, Nov 2011. Available at: `http://www.w3c-test.org/webperf/specs/NavigationTiming/`. Accessed 11-Febuary-2012.

[44] Wium Lie, H., Çelik, T., Glazman, D., and van Kesteren, A. Media Queries. W3C Candidate Recommendation, W3C, Jul 2010. Available at: `http://www.w3.org/TR/css3-mediaqueries/`. Accessed 7-Febuary-2012.

[45] Zakas, N. C. *High Performance JavaScript*. O'Reilly Media / Yahoo Press, 2010.