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 29th April, 2012 — DRAFT!

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

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:**	ix + 70
Professorship:	Media Technology	**Code:**	T-111
Supervisor:	Professor Petri Vuorimaa		
Instructor:	Risto Sarvas D.Sc.(Tech.)		

In twenty years, the Web has become an integral part of our everyday lives. The rapid growth of the smartphone market has brought the Web from our home desks to anywhere we are, and enabled us to access this vast source of information at any time.

However, the proliferation of mobile devices and platforms has raised new problems for application development. The growing amount of different platforms and their distinct native technologies make it hard to develop applications that can be accessed with all these devices.

The only combining factor in all these platforms is the browser, and it is becoming the universal application platform. We cannot afford anymore to build applications for the silos and walled gardens of single platforms, and building cross-platform applications is essential in the modern mobile market.

In this work, I introduce the HTML5 (HyperText Markup Language version 5) specification as well as several related specifications or specification drafts for modern web development. I also present several tools and libraries for mobile web development.

I implemented a mobile web application and a network utility library, and assessed the practical performance of the modern tools and APIs (Application Programming Interface). In this work, I present the tools and techniques for performance optimization of mobile web applications.

Keywords:	mobile, HTML5, cross-platform, performance
Language:	English

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:**	ix + 70
Professuuri:	Mediatekniikka	**Koodi:**	T-111
Valvoja:	Professori Petri Vuorimaa		
Ohjaaja:	Tohtori Risto Sarvas		

Kahdenkymmenen vuoden aikana webistä on tullut oleellinen osa jokapäiväistä elämäämme. Mobiilimarkkinoiden huikea kasvu on tuonut webin kotipöydiltämme mukaamme missä ikinä olemmekin ja mahdollistanut tämän laajan tietovaraston käyttämisen milloin tahansa.

Mobiililaitteiden käytön räjähdysmäinen kasvu on kuitenkin nostanut uusia haasteita ohjelmistokehitykselle. Monien eri alustojen natiiviteknologiat poikkeavat toisistaan, ja ohjelmistojen kehittäminen kaikille näille alustoille on haastavaa.

Ainoa yhteinen tekijä näissä alustoissa on WWW-selain (World Wide Web), josta on tulossa universaali ohjelmistoalusta. Enää ei voida kehittää ohjelmistoja vain tiettyjen suljettujen alustojen käyttäjille, ja alusta-riippumattomuudesta on tullut oleellinen osa mobiilimarkkinoita.

Tässä työssä esittelemme HTML5-standardin sekä muita siihen liittyviä standardeja sekä standardiluonnoksia, jotka tuovat uusia ominaisuuksia ja helpotuksia web-kehitykseen. Esittelemme myös useita työkaluja ja tekniikoita moderniin web-kehitykseen mobiililaitteille.

Toteutimme mobiililaitteissa toimivan web-ohjelmiston sekä kirjaston tiedon siirtämiseen mobiiliverkoissa, ja arvioimme modernien työkalujen ja rajapintojen käytännön suorituskykyä. Tässä työssä esitämme useita työkaluja ja tekniikoita web-ohjelmistojen suorituskyvyn optimointiin mobiililaitteille.

Asiasanat:	mobiili, HTML5, alusta-riippumattomuus, suorituskyky
Kieli:	Englanti

Acknowledgements

I would like to thank my supervisor, Professor Petri Vuorimaa, for the valuable comments and instructions for this work; my instructor, Doctor Risto Sarvas, for giving me a valuable point of view to see this work in the proper context as a whole; and my colleague, Jarno Rantanen, for making the demo application for the utility library documented in this work. I would also like to thank my employer, Futurice, for giving me time to write this work and to do satisfying projects, such as the practical part of this work, every day with different customers.

Mostly, I would like to thank my wife for the valuable support and my firstborn son, Leo, who was born during the writing of this work. You are the light of my life.

Kirkkonummi, !Fixme **Add English date** Fixme!

Kimmo Puputti

Abbreviations and Acronyms

3G	3rd Generation Mobile Telecommunications
API	Application Programming Interface
Ajax	Asynchronous JavaScript and XML
AppCache	Application Cache
Blob	Binary Large Object
CDN	Content Delivery Network
CORS	Cross-Origin Resource Sharing
CPU	Central Processing Unit
CSS	Cascading Style Sheets
CSS3	Cascading Style Sheets level 3
DNS	Domain Name System
DOM	Document Object Model
ETag	Entity Tag
GPS	Global Positioning System
HTML	HyperText Markup Language
HTML5	HyperText Markup Language version 5
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
IT	Information Technology
JSON	JavaScript Object Notation
MVC	Model-View-Controller
MathML	Mathematical Markup Language
OS	Operating System
OpenGL® ES	Open Graphics Library for Embedded Systems
REST	Representational State Transfer
RTMP	Real Time Messaging Protocol
SDK	Software Development Kit
SQL	Structured Query Language
SVG	Scalable Vector Graphics

TCP	Transmission Control Protocol
TTL	Time To Live
UA	User-Agent
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WHATWG	Web Hypertext Application Technology Working Group
WWW	World Wide Web
WebGL	Web Graphics Library
WiFi	Wireless Local Area Network (WLAN)
XHTML	eXtensible HyperText Markup Language
XML	Extensible Markup Language

Contents

Abbreviations and Acronyms **iv**

1 Introduction **1**
1.1 Research Questions 4
1.2 Structure of This Work 4

2 HTML5 **5**
2.1 Semantic Markup 6
2.2 Extensibility 7
2.3 Media 8
2.4 Canvas 2D Context 8
2.5 Form Enhancements 8
2.6 Session History Manipulation 9
2.7 Offline Web Applications 9
2.7.1 Application Cache 9
2.7.2 Data Storage 10
2.7.3 Detecting Network State 11
2.8 Drag and Drop 11
2.9 SVG and MathML 11

3 Other Related Specifications **13**
3.1 Cascading Style Sheets 13
3.2 WebGL and Typed Arrays 15
3.3 Touch Events 16
3.4 Files 16
3.5 Web Real-time Communication 16
3.6 Web Sockets 17
3.7 Server-Sent Events 18
3.8 Web Workers 18
3.9 Analytics and Timing 19
3.10 Page Visibility and Timer Control 19

3.11 Cross-Origin Resource Sharing 20
3.12 Device APIs 20
3.12.1 Geolocation 20
3.12.2 Device Orientation 21
3.12.3 User Media 21
3.13 Other 21

4 Tools for Modern Mobile Web Applications 23
4.1 Single-Page applications 24
4.2 JavaScript MVC Libraries 24
4.3 Responsive Design 26
4.4 Progressive Enhancement 26
4.5 User Interface Libraries 27
4.5.1 jQuery Mobile 27
4.5.2 jQTouch 27
4.5.3 Sencha Touch 29
4.6 Hybrid Applications 29
4.7 Performance Guidelines 30

5 Use Case 36
5.1 Conference Application Requirements 36
5.2 JSONCache Requirements 37

6 Implementation and Results 38
6.1 Conference Application 38
6.2 JSONCache JavaScript Library 40
6.3 Targeting Different Platforms 42
6.3.1 Device Detection 43
6.3.2 Feature Detection 44
6.4 Targeting Different Screens 45
6.5 Handling Different Orientations 47
6.6 Handling Mobile Networks 49
6.6.1 Minimizing Data Transfer 49
6.6.2 Caching 49
6.6.3 Preloading 50
6.6.4 Offline Support 50
6.6.5 Handling Interruptions 51
6.7 Animations 51
6.8 Following JavaScript Best Practices 52
6.8.1 JSLint 52
6.8.2 Lazy initialization 52

6.8.3 Efficient DOM Manipulation 53
6.8.4 Efficient Event Handling 53
6.9 Performance Analysis 55
6.9.1 YSlow 55
6.9.2 Page Speed 55
6.10 Summary of Results 59

7 Conclusions **62**
7.1 Further Work 62
7.2 Discussion 63

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. [15]

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. [15]

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. [15]

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 [9]

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. [9, 10, 15]

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. [9]

Table 1.1 shows the required developer skill sets for different platforms. As we see, each platform has its own programming language and SDK (Software Development Kit). 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. [9]

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 (see smartphone sales by operating system in Figure 1.1). [4]

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. [43, 55]

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

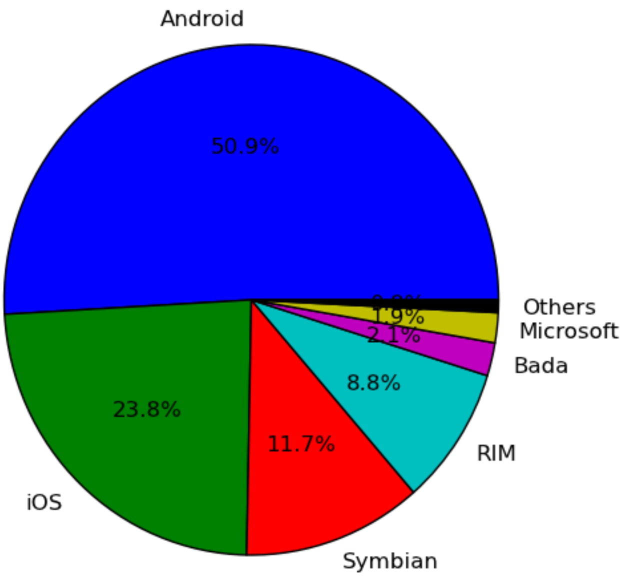

Figure 1.1: Worldwide Smartphone sales by operating systems in 4Q 2011 according to `http://www.gartner.com/it/page.jsp?id=1924314`

for sharing documents into a massively popular, world wide application and information distribution environment [55]. During the so-called Web 2.0 revolution, the Web grew into a platform for interactive applications with the help of technologies like Ajax (Asynchronous JavaScript and XML) [19].

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 [9]. By performance, I 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. [9, 15]

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

1.1 Research Questions

Knowing the reasons and motivation for cross-platform HTML5 and the importance of application performance and responsiveness:

- **RQ1**: *What are the main problem areas in mobile web development?*

  Mobile web development is a large problem area, and dealing with relatively new technologies and large amounts of devices, finding the main problems is crucial for this work.

- **RQ2**: *Do HTML5 and related specifications solve these problems?*

  There are a lot of new specifications and APIs for web development. Do these specifications solve the problems identified in answering RQ1?

- **RQ3**: *What other practical means do we have to solve these problems?*

  Not all problems can be solved with specifications and new standard APIs. What other practical means and techniques can be used to solve the problems identified in RQ1?

I introduce the HTML5 specification and other related standards in a generic way, but the practical research is constrained into using the new APIs in a real-world mobile application. Thus not all APIs are applicable or needed, but the ones used are deployed for real end users and examined in the current browser implementations. Being a large topic, this work focuses especially on the performance aspect of the latest APIs and their current implementations to get a good overview of the application of the specifications in a real-world scenario.

1.2 Structure of This Work

In Chapter 2 and Chapter 3 I introduce HTML5 and related specifications and APIs for modern web development. In Chapter 4 I present some of the latest tools, frameworks and libraries for building mobile web applications.

In Chapter 5 I introduce the practical part of this work and its requirements, and in Chapter 6 I present the implementation details and results of the practical research. In Chapter 7 I sum up the work and discus further work ideas.

Chapter 2

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. [46]

In the early 2000s, W3C (World Wide Web Consortium) was developing XHTML (eXtensible HyperText Markup Language) 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. WHATWG (Web Hypertext Application Technology Working Group) defines itself as a “loose, unofficial, and open collaboration of Web browser manufacturers and interested parties”[1]. [46]

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 vendors 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

[1] `http://www.whatwg.org/news/start`

[2] `http://dev.opera.com/articles/view/mama-key-findings/`

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. [46]

In this work, I 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 my opinion, the web development community has on HTML5.

2.1 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 a more semantic structure of HTML by dropping many presentational features. The rationale behind this is explained with the following reasons [23]:

- 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 [23]:

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

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

[3] `http://code.google.com/webstats/`

- **Flow content:** Most content that is 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`

2.2 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. [23]

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

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

2.3 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.

2.4 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. [23]

The Canvas 2D Context specification draft [24] 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.

2.5 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 reduce the need for scripting. [23]

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.

2.6 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 (Uniform Resource Locator). 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 and forth 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. [23]

2.7 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 [65], which has raised the need for offline support in HTML5.

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

2.7.1 Application Cache

AppCache (Application Cache) 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 as well as fallback URLs for resources that are not cached but with which the fetching fails. In addition to the simple manifest file listing offline resources, JavaScript events are defined for cache changes. [23]

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
```

2.7.2 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. [28]

Two more expressive storage APIs have been specified: client side SQL (Structured Query Language) database [28] and the Indexed Database [42]. The client side SQL database defines an asynchronous and transactional JavaScript API for a SQL database. 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 querying 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 relative complexities of SQL.

2.7.3 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. [23]

Although providing important information, these network status indicators are inherently unreliable [23]. 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 unreliable.

2.8 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 (Document Object Model) 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.

2.9 SVG and MathML

While not part of the HTML5 standard, the specification allows for embedding SVG (Scalable Vector Graphics) [1] and MathML (Mathematical Markup Language) [8] markup within HTML.

SVG is a markup language for describing two-dimensional vector graphics in XML (Extensible Markup Language). 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 to fit the available space no matter what the screen size is, which improves the cross-platform usefulness for different device form factors.

Chapter 3

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 (Cascading Style Sheets), and JavaScript in modern web applications. In the following sections, I introduce the specifications that are of interest within the topic of this work.

3.1 Cascading Style Sheets

CSS Level 3 specifications introduce lots of new functionality for web application styling. Well separated layout layer keeps the document structure clean, and rich styling and effects capabilities reduce the need for scripting and provide graceful fallback functionality for older user agents. By letting the browser handle, for example, rich user interface animation effects allows developers to easily optimize the responsiveness and performance of their applications since the browser can use the most efficient techniques of the platform to handle these effects. Below I list the main components and specifications of the W3C CSS working group[1].

- **Selectors**

  CSS selectors are patterns that are used to match elements in a DOM tree. The patterns can then be used to apply style rules to the matched

[1] `http://www.w3.org/Style/CSS/members.en.php3`

elements. Selectors can also be used in JavaScript to select elements for scripting.

CSS3 (Cascading Style Sheets level 3) defines a set of new selectors [66] for powerful matching of elements in complex DOM trees. These selectors are useful for rich interactive web applications and reduce the need for scripting for element matching. Efficient selectors are also important for performance optimization.

- **Transforms**

  2D and 3D transforms [17] allow elements to be transformed in two-dimensional or three-dimensional space. Elements can be translated, rotated, and scaled in their coordinate space. The specification defines several 2D and 3D transformation functions, which can be used in the transforms.

  Transforms can provide subtle but important user interface effects, or they can be used for advanced interactive graphics. Combined with, for example, timed animations or transitions, rich user interfaces can be built with declarative CSS rules.

- **Transitions**

  CSS transitions [30] allow element styles to change smoothly over a specified duration, and they can be used for simple animations. Normally when the value of a CSS property changes, the result is seen instantly, but with transitions, the changes can be timed and configured for presentational effects.

- **Animations**

  Simple animations between two layout states can be done with transitions, but for more complex series of changes, CSS animations [31] can be used.

  The animations specification defines so-called keyframes, which can be used to specify the progress of the animation between the start and the end states. Animations can also be configured to repeat a certain number of times, to alternate between the begin and end values, to control the running and paused states, and to delay the start time. [31]

- **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. [63]

Media queries can be used to detect the device screen size and dimensions, orientation, aspect ratio, color depth, etc. [63]. 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.

- **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.

  CSS3 defines techniques to dynamically load custom fonts and specify their properties [12]. 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.

3.2 WebGL and Typed Arrays

WebGL (Web Graphics Library) is a low-level 3D rendering API derived from the OpenGL® ES 2.0 (Open Graphics Library for Embedded Systems) [44] 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 a global platform for cross-platform games. WebGL was originally developed by the Khronos Group[2] and later the specification work was participated by browser vendors and 3D developers. [41]

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 their own specification [22]. Traditional JavaScript arrays are not typed, but due to performance reasons, WebGL API needed more efficient

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

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.

3.3 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 [6] 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.

3.4 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 [48, 57, 58] 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 3.8). Text and binary files can be manipulated in memory or as Blob (Binary Large Object) 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 2.8) and Web Workers (See Section 3.8), HTML5 forms can be greatly enhanced and optimized with richer interactivity and better performance.

3.5 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[3] and RTMP[4] (Real Time Messaging Protocol), 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 [7] (see Section 3.12.3), 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.

3.6 Web Sockets

Web Sockets API [13, 27] defines a two-way communication protocol for real-time applications between a client, such as web browser, and a remote server. Because HTTP (Hypertext Transfer Protocol) 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 (Hypertext Transfer Protocol Secure). The API uses a single TCP (Transmission Control Protocol) connection that is kept open and allows for traffic in both ways. [13, 27]

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 (Internet Protocol) 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. [13]

[3] `http://www.adobe.com/products/flashplayer.html`
[4] `http://www.adobe.com/devnet/rtmp.html`

3.7 Server-Sent Events

Server-Sent Events specification [26] 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.

3.8 Web Workers

The whole IT (Information Technology) 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 (Central Processing Unit). 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 therefore 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. [52]

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. [29]

Web Workers are external JavaScript files initialized from a web site. The worker runs in a separate OS-level (Operating System) thread and does not affect the main user interface thread apart from the communication channel.

The simple API is easy to work with and enables the long-needed ability of parallel computation in web applications.

3.9 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 [62] 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 [38] and general purpose user action profiling [39]. The Performance Timeline specification [37] defines a unifying interface to access these performance metrics.

3.10 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 [36] 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 [49] and the Efficient Script Yielding specification [40] 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.

3.11 Cross-Origin Resource Sharing

Same-origin policy is an important security restriction in web browsers preventing web applications to obtain data from other origins [59]. 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 (CORS) specification [59] 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 easier and safer.

Cross-domain resource usage can also be controlled with the means defined in the Content Security Policy specification [54] and The From-Origin Header specification [60]. The specifications define HTTP headers that can be used to restrict certain content types to be allowed only from the given origins. In addition to these specifications, the HTML5 Web Messaging specification [25] defines an API for cross-document messaging.

3.12 Device APIs

One of the main differences between native and web applications is the device sensor access. In the following sections I present some specifications that define APIs to access the device features from JavaScript.

3.12.1 Geolocation

Context is one of the main components of a personalized application [15]. 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 (Global Positioning System) chip on a mobile device or a WiFi (Wireless Local Area Network, WLAN) network with a known location. The coordinates and their accuracy are provided by the API. [47]

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.

3.12.2 Device Orientation

Knowledge of the physical orientation of a device and changes in the orientation enable, for example, highly interactive games with rich input mechanisms. Typical sensor sources for the orientation include gyroscopes, compasses, and accelerometers [5].

The Device Orientation specification [5] defines APIs and events to access this sensor data through an abstraction layer. The data can be used for rich interaction and gestures for a wide array of applications.

3.12.3 User Media

The getusermedia specification [7] defines means for accessing multimedia streams from local devices. The streams can be audio, video, or both, and the source device can be, for example, the web camera on a desktop computer or the microphone on a mobile device.

The `getUserMedia` function can be used to request access to a multimedia stream, and the stream can then be handled, for example, with the File API (see Section 3.4) for recording to a file, or used as a source in an `audio` or a `video` element.

3.13 Other

There are also lots of other specifications, drafts, and proposals for other device feature and sensor data access. Below I list some of these.

- **Web Notifications** [21] for displaying simple notification to the user
- **Fullscreen** [61] for controlling fullscreen display of the application
- **Pointer Lock** [50] for controlling input pointers

- **Clipboard** [53] for using copy, cut, and paste operations
- **Gamepad** [20] for a low-level interface to gamepad devices
- **Battery Status** [33] for accessing information about the battery status of the device
- **Vibration API** [32] for accessing the vibration mechanism of the device
- **Network Information** [35] for an interface to access the underlying connection information or the device
- **Contacts** [56] for an interface to access the address book of the user
- **Web Intents**[5] for service discovery and inter-application communication.

[5] http://webintents.org/

Chapter 4

Tools for Modern Mobile Web Applications

With all the hype and trending surrounding HTML5 and mobile web applications, several libraries and tools have been developed to help building these applications. Fling [16] represents a typical anatomy of a mobile HTML5 application as shown in Figure 4.1.

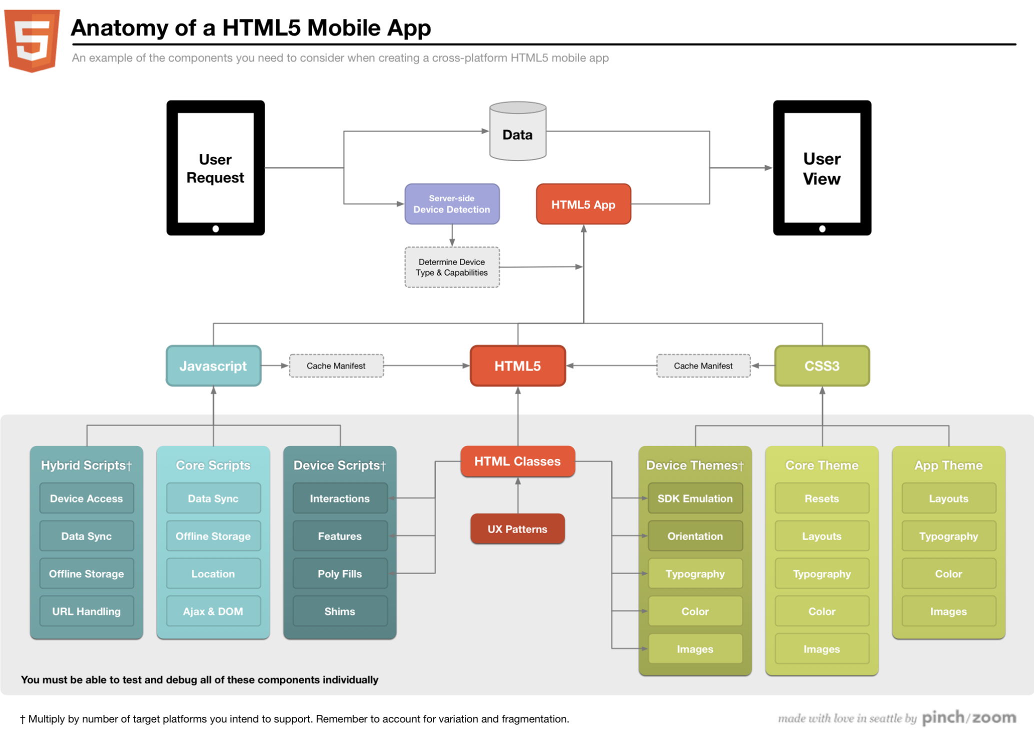

Figure 4.1: HTML5 Mobile Application Anatomy according to [16].

In this chapter, I present tools, libraries, and techniques for building modern web applications.

4.1 Single-Page applications

In recent years, the movement from traditional interlinked documents to interactive web applications has had a profound effect on the architecture of web applications. Traditional web sites have structured backend architectures often with a database layer, a layer for business logic, and a layer for generating the HTML documents from a template. The frontend usually uses CSS for layout and styling and some JavaScript for enhancing forms or some interactive components on the page.

These conventional three-tiered web applications [34] often use an MVC (Model-View-Controller) [18] framework for separating the data, logic, and presentation layers in the backend. However, the latest trend in web application development has been having only a simple REST (Representational State Transfer) [14] API as a backend data layer, and using a JavaScript MVC framework in the frontend. Thus the whole application logic and presentation has moved to the client side, with the backend only working as a data persistency layer.

Using modern JavaScript storage APIs (see Section 2.7.2), even the data persistency and application state layer can be in the client side, with backend only working as an application and data delivery layer. In addition to the client side data persistency, the storage APIs can be used to store the user-specific application state.

Single-page applications might introduce bigger initial page load, but after the startup, the network is only used when interacting with the backend data API. This makes the applications faster and more responsive, and minimizes the effects that unreliable networks have on the application usage since separate page views do not require a new document request.

4.2 JavaScript MVC Libraries

Due to the recent trend to move application logic from the backend to the frontend, JavaScript code bases have grown into large applications that need a proper modular structure. Several frameworks have been developed to structure JavaScript applications into well-separated modules and layers.

Many of the JavaScript application frameworks are derived from the MVC architecture pattern adapting it to the design needs and requirements of browser-based applications. Example frameworks include Backbone.js[1], Spine[2], ember[3], batman.js[4], Knockout[5], and JavaScriptMVC[6].

One popular framework nowadays is Backbone.js, which I also chose for the application described in Chapter 5. Backbone.js is an open source JavaScript framework providing the essential components and structures for building large JavaScript applications. Backbone.js provides the following components:

- **Model**

  Models provide the domain-specific data layer of the application. They provide data manipulation, persistency, and serialization methods as well as an event handling mechanism for data changes.

- **Collection**

  Collections are ordered sets of models. They can be used to observe and manipulate models as a group. They can also be used to filter specific models for some purposes.

- **Router**

  Routers provide methods for routing between pages of an application by observing and modifying the URL. URLs can be mapped to events and actions for client-side application navigation.

- **History**

  The History utility is used together with Routers to handle the application navigation to preserve the back button functionality and the bookmarking of certain pages in the application.

- **Sync**

  The Sync utility provides data synchronization to the backend.

- **View**

  Views are structures that help organizing the user interface into logical parts. They usually observe certain models or collections for changes,

[1] `http://backbonejs.org/`
[2] `http://spinejs.com/`
[3] `http://emberjs.com/`
[4] `http://batmanjs.org/`
[5] `http://knockoutjs.com/`
[6] `http://javascriptmvc.com/`

and update themselves independently of each other when the underlying data changes. They are often used together with some templating library, such as Mustache[7] or Handlebars[8].

4.3 Responsive Design

Responsive design is a way to design a web page to fit to varying sizes of screens and devices. The traditional way to design a web page is to compromise on a certain width based on the expected desktop screen sizes of the target audience and to lay out the elements of the page to the chosen width.

Using Media Queries (see Section 3.1), we can provide tailored layouts for different screen sizes. For example, we can swap the images to smaller ones for mobile devices or hide some elements to make the layout cleaner on small screens. This enables us to use the same code base to target all devices and screens.

4.4 Progressive Enhancement

Modern web sites and web applications are accessed with a huge variety of devices and form-factors with varying capabilities, and supporting all the possible browsers your users might have becomes a huge burden on developers. New standards support and APIs in latest browsers seem tempting and valuable, but having to support also less-capable browsers prevent developers from using a single solution for all browsers.

The goal of progressive enhancement is to provide universal access to a web site or an application no matter what capabilities the browser or the user has. Parker et al. define the three key principles of progressive enhancement [45]:

- Start with clear content and well-structured markup.
- Maintain strict separation of layout and presentation.
- Unobtrusively layer in advanced behavior and styling, with careful consideration of accessibility implications.

Often the design is started "mobile first" meaning that the simplest and the most universal bottom layer of the application is designed for the least-capable browsers with very little screen estate. This forces the design to be

[7]http://mustache.github.com/
[8]http://handlebarsjs.com/

simple and semantic, and filters out any extra markup that is not needed for the semantic presentation of the content and functionality. In addition, by designing first to browsers that might not even have CSS support forces a clear separation of layout and content as well as makes the clean markup easier to style and enhance [45].

Using feature detection (see Section 6.3.2), more layers are added on top of the clean markup. The objective is to use unobtrusive JavaScript to enhance the markup to avoid breaking parts of the page or the whole site with careless scripting. By using feature detection, we ensure that we take the most out of the latest browsers by using their full capabilities, and at the same time, keeping the application accessible and functional in less-capable browsers. This also makes the applications future-proof when browsers are updated and new APIs and features are implemented in them. [45]

4.5 User Interface Libraries

User Interface libraries help in developing mobile web applications fast by providing finished and tested widgets and user interface components that can be combined and configured easily. I present some popular frameworks in the following sections.

4.5.1 jQuery Mobile

jQuery Mobile[9] is a client side framework optimized for touch devices. The user interface is based on HTML5 and the jQuery JavaScript framework[10]. The aim of the project is to provide a progressively enhanced web framework for as many devices as possible. Figure 4.2 shows example components in the framework.

jQuery Mobile is an open source project sponsored by large mobile and media companies. The user interface is fully theamable and there are several third-party extensions and widgets to the framework.

4.5.2 jQTouch

jQTouch[11] is a lightweight open source library for high-end smartphones and tablet devices. It only supports the WebKit browser engine [12] used in iOS,

[9] http://jquerymobile.com/
[10] http://jquery.com/
[11] http://jqtouch.com/
[12] http://www.webkit.org/

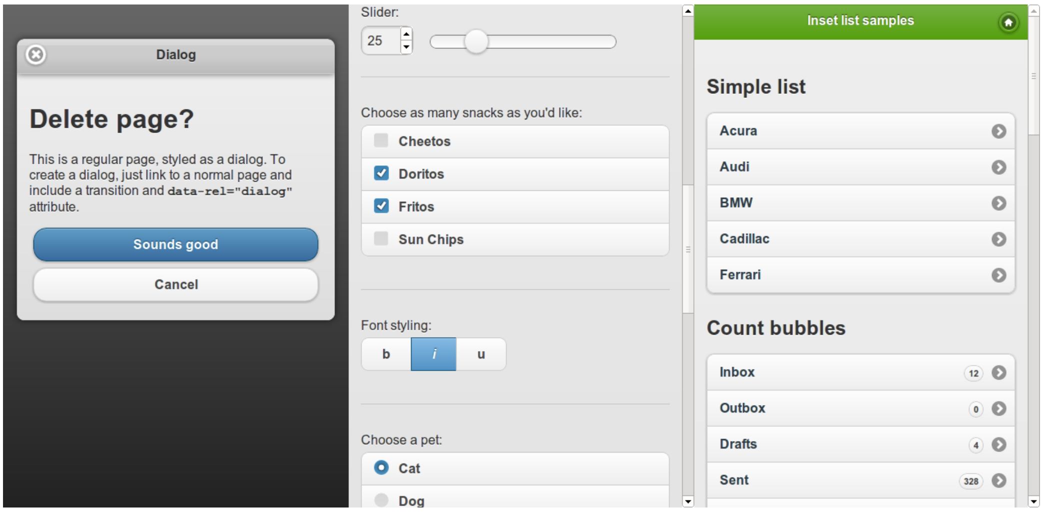

Figure 4.2: jQuery Mobile user interface components. `http://jquerymobile.com/demos/1.0.1/`

Android, Blackberry, and WebOS devices. It provides customizable themes and user interface components, as well as helpers for handling touch input. Figure 4.3 shows example components of the library.

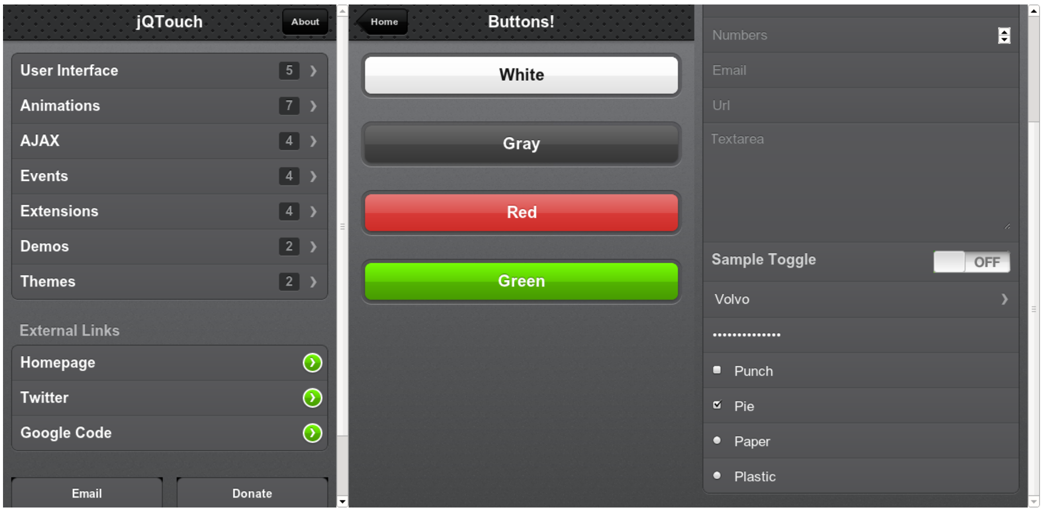

Figure 4.3: jQTouch user interface. `http://jqtouch.com/preview/demos/main/`

4.5.3 Sencha Touch

Sencha Touch[13] is an open source HTML5 mobile web application framework for iPhone, Android, and Blackberry devices. The framework is fully theamable and comes with a large set of user interface components. It also provides helpers for touch input handling. Figure 4.4 shows example components of the library.

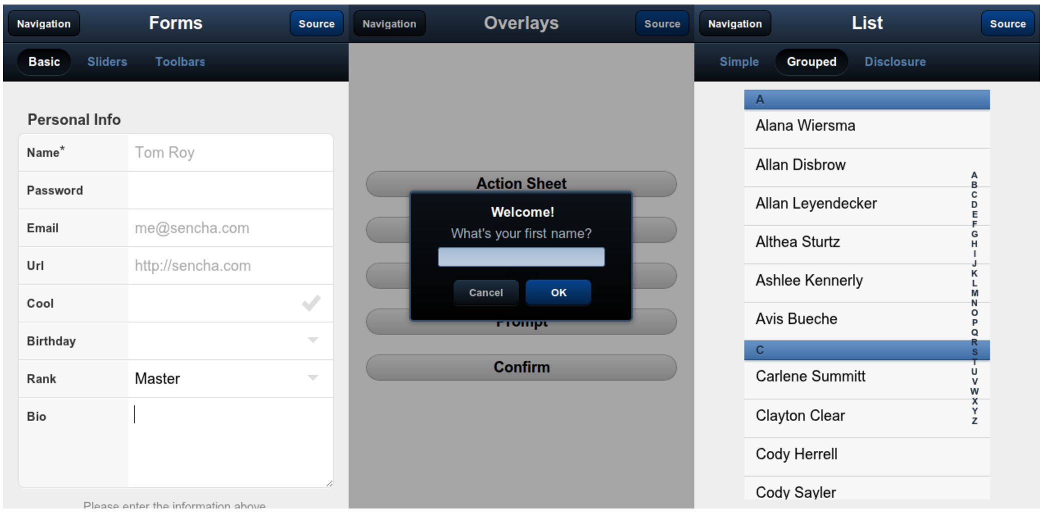

Figure 4.4: Sencha Touch user interface. `http://dev.sencha.com/deploy/touch/examples/kitchensink/`

4.6 Hybrid Applications

Sometimes application stores can be a valuable marketing path, and visibility in these stores can bring a lot of users to an application. The stores also provide easy billing of applications themselves, and solutions for in-app billing. [10]

Fortunately, modern smartphone and tablet platforms provide a user interface component to embed a web browser view into a native application. This web view component can be used to build parts or even the whole application with web technologies. For example, the web view can take the whole available screen space of the application, and show a local HTML document together with local assets like CSS and JavaScript files and images.

[13] `http://www.sencha.com/products/touch`

These applications that use a native web view wrapper with parts of the application built with web technologies are called hybrid applications. The level of native versus web technologies can vary, and everything is possible from fully utilizing web technologies with just a simple native wrapper to a native application with just a simple static HTML document.

Several tools have been developed to build hybrid applications. One of these is PhoneGap [14], which is an open source HTML5 application platform for building native applications with web technologies. PhoneGap provides native wrappers for all of the largest mobile platforms, and exposes extra JavaScript APIs that are not accessible in normal web pages. These APIs are based on the latest specifications, and they can be used to access device functionality that has previously been inaccessible using JavaScript.

With tools like PhoneGap, developers can build native applications using familiar web technologies with access to native features and device APIs. These hybrid applications have only one code base that is deployed for several smartphone platforms, and the applications can be submitted to application stores.

4.7 Performance Guidelines

There are several web application performance best practices and related guidelines. According to Souders [51], 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 I list the performance guidelines defined by Souders [51].

- **Make Fewer HTTP Requests**

  According to Souders, 80–90% of the end user response time is spent downloading components on 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 reduce the number of needed HTTP requests. Combining images into sprites, inlining images, or combining separate JavaScript and CSS files results in fewer components needed to download on a page.

[14] `http://phonegap.com/`

- **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 an HTTP request altogether is the best option for reducing the response time when downloading the components on 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 tells the client how long a resource is valid, and especially far future Expires headers reduce the need for downloading and updating the components on 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 to 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 (Domain Name System) 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 (Time To Live) 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 variables and function names to shorter ones without changing the interpreted behavior of the code.

- **Avoid Redirects**

  Rerouting any component on 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 (Entity Tag) 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 [19] functionality, and developers should make sure all the requested URLs for data fetching follow the performance best practices such as having proper caching in place.

In addition to the performance rules [51], Souders also specifies additional techniques to improve performance [52]:

- **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 to 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 an 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 user interface 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 user interface from updating and can result in bad user experience.

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. [64]

- **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 5

Use Case

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

In this chapter I present the implementation requirements for the conference application as well as the requirements for the JSONCache network utility library.

5.1 Conference Application Requirements

The target group for the conference application was mobile developers attending the Qt Developer Days conference. Therefore, I could expect a good technical knowledge and high-end mobile devices from the target audience.

A native version of the application was built for devices with Qt support, and the HTML5 application was for all other devices. The main target devices were iPhone, Android devices, Windows Phone 7 devices, and iPad. In addition to these, the application was tested on devices running Symbian and Meego, as well as desktop browsers.

The conference was expected to have some thousands of attendees, of which a few hundred was expected to use the web application. In conferences of this size, network connectivity and reliability is often a problem. Also, mobile networks other than the WiFi network supplied by the conference might be too expensive for users that come from other countries. This is why offline support was needed.

[1] `http://qt.nokia.com/qtdevdays2011/`
[2] `http://qt.nokia.com/`

The client wanted high interactivity and personalization in the application. Users could save interesting sessions to their favorites, which were shown in the home view of the application. The home view was expected to be contextual in taking into account the current time and showing the ongoing sessions and the remaining time for them, as well as the time left for later favorite sessions to start. Current time was also expected to be indicated in the agenda view, where a red line was to be drawn to the current time for easily visualizing the ongoing sessions. By default, the agenda view should show the ongoing day of the conference if possible.

The user interface was required to use the touch input interactions for panning and zooming the floor maps of the conference venue. Also, the client wanted a touchable star rating widget on the feedback form for easy and visual session rating on touch screens.

To sum up, the main requirements are as follows:

1. Cross-platform support for high-end touch screen smartphones and tablets
2. Flexible design and layout to take the available screen into account
3. Personalized (session favorites) and contextual experience taking the current date and time into account
4. Offline support
5. Floor map view with pinch-to-zoom gesture and panning support
6. Rich agenda and track view with sessions shown visually in a timeline.

5.2 JSONCache Requirements

JSONCache was designed as a JavaScript network utility library for unreliable mobile networks. It is used in Ajax data requests with JSON (JavaScript Object Notation) data. The main idea was to avoid refetching data that was already fetched and to handle network interruptions without the user noticing anything. The requirements are as follows:

7. Cache data in the client side (localStorage) to avoid fetching data that has already been transferred
8. Attempt to fetch the data multiple times if the requests fail for some reason.

Chapter 6

Implementation and Results

I implemented two example cases: a mobile web application for a developer conference and a networking library for data caching and fetching multiple times. Here I present the architecture, components and the implementation details I used for fulfilling the requirements listed in Chapter 5 and the techniques I used and the compromises I had to make to tackle the problems in mobile web development. I used tools introduced in Chapter 4 and APIs introduced in Chapter 2 and Chapter 3.

6.1 Conference Application

The conference schedule[1] is a single-page application (see Section 4.1) with a lightweight backend written in Python using the Django Web Framework[2].

The backend provides the static assets (JavaScript, CSS, images, etc.) and an API for persisting session feedback to a MySQL[3] relational database. It also generates the HTML5 AppCache (see Section 2.7.1) offline cache manifest file based on the categorized device type.

The frontend is a JavaScript application written using the Backbone[4] MVC framework (see Section 4.2). Other JavaScript libraries include Underscore[5] for data manipulation, jQuery[6] for DOM API abstraction, Handlebars[7] for templating, and Modernizr[8] for feature detection. The HTML5 Mobile

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

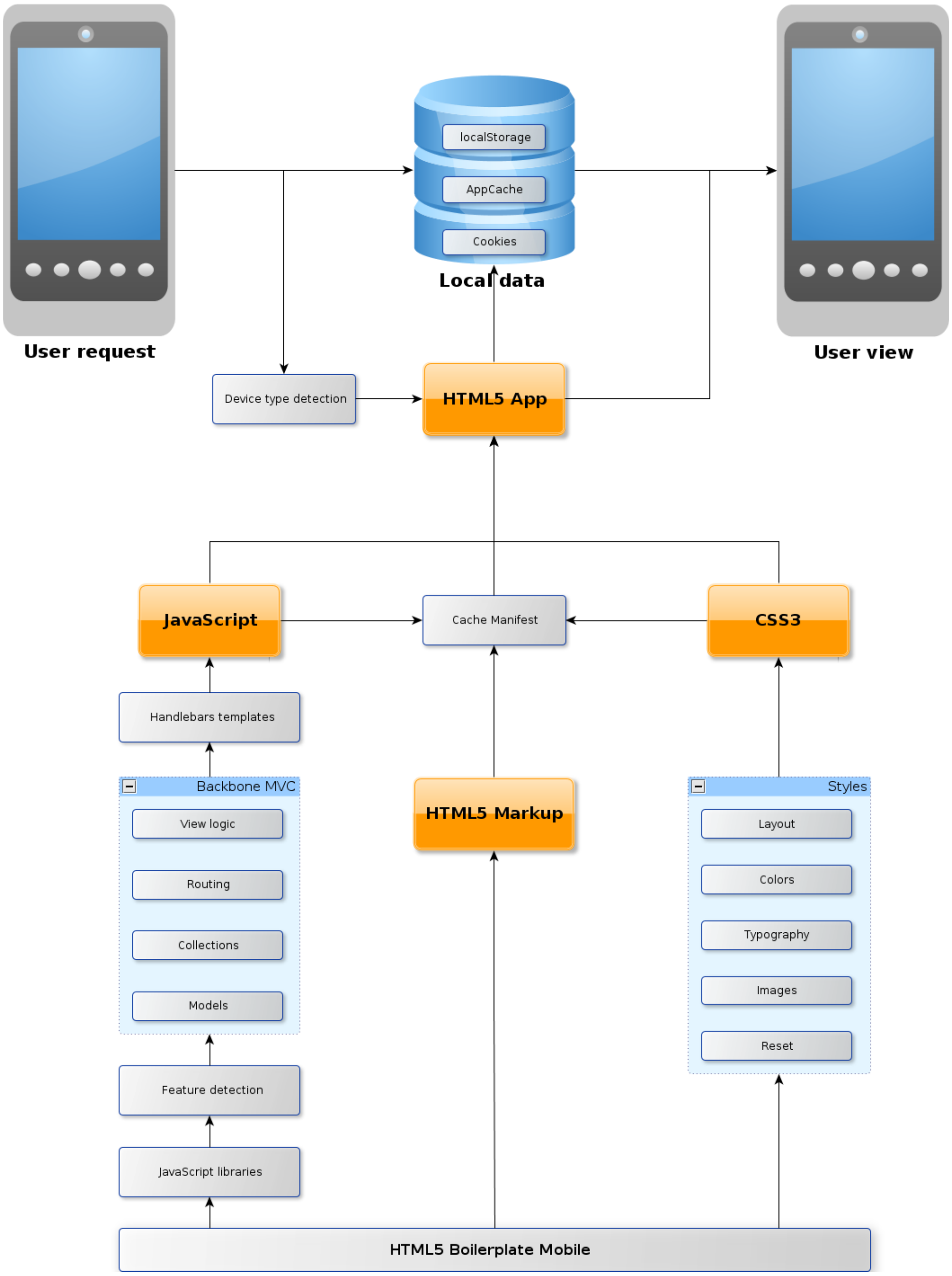

Figure 6.1: Conference schedule application architecture.

Boilerplate[9] was used as an initial markup structure for the application. The architecture of the application is shown in Figure 6.1. 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. Figure 6.2 shows an example view in the application.

6.2 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 connection is 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 (see Section 2.7.2) 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 is not expired, it is returned immediately. If the data is not in the cache or it is 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. For example, when a user leaves his or her work place and uses a web application, the mobile device changes from the workplace WiFi network to the 3G (3rd Generation Mobile Telecommunications) network, there is a short interruption in the connection, and any ongoing network requests are affected.

If a data fetch fails, a new fetch is issued after a timeout (defined in the configuration). On subsequent attempts the timeout is increased, and after a defined number of attempts the fetch error is issued.

Figure 6.3 shows an interactive demo of the JSONCache library. The demo[10] simulates the caching and fetching functionality of the library by

[9] `http://html5boilerplate.com/mobile`

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

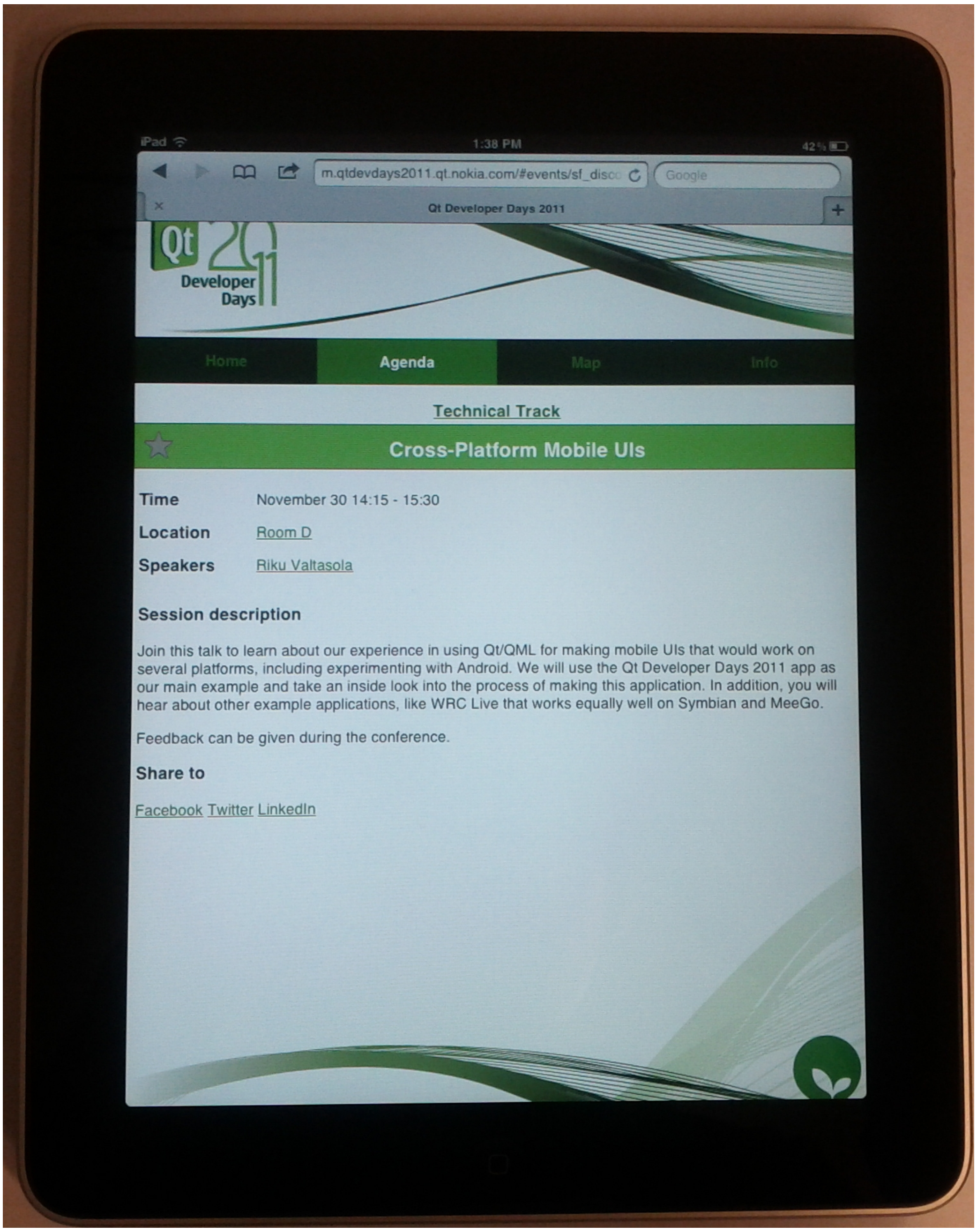

Figure 6.2: Session details view on an iPad.

demonstrating an unreliable network based on the configuration.

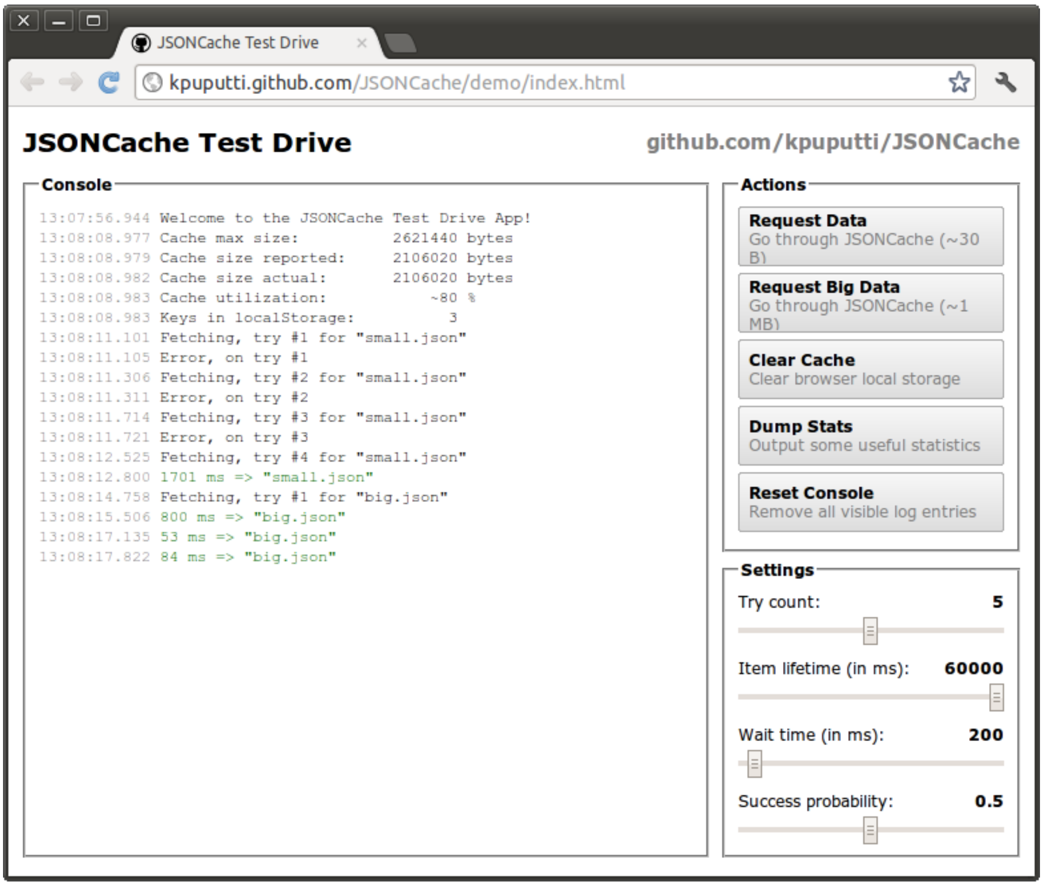

Figure 6.3: Interactive JSONCache demo.

6.3 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. Browsers also have known bugs that have to be handled.

Therefore, means to detect the user's device are needed (see requirements 1 and 2 in Section 5.1). Here I present two such means: device detection and feature detection. Both of these were used in our conference application.

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 6.1: Example User-Agent strings.

6.3.1 Device Detection

The User-Agent (UA) HTTP header contains detailed information of the web browser and the platform where the request originates from. As can be seen from Table 6.1, 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 (see Section 2.7.1) 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 6.2. 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. I could have listed all the assets for all the devices, but then the list of offline 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 out-

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 6.2: Device type detection rules.

weighed the possible compromises. The worst that could happen was that the device was wrongly classified and some graphics assets 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 providing 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 browser and platform updates, and the application might break due to invalid expectations. This is why feature detection is generally the recommended option whenever possible.

6.3.2 Feature Detection

Feature detection is an important concept in designing with progressive enhancement (See Section 4.4). A lot of the HTML5-related JavaScript APIs are still unsupported in several platforms, but browser developers are constantly filling in the gaps. Therefore, it is important to check whether a certain feature is supported and to provide graceful fallback mechanisms for browsers lacking the functionality.

Doing run-time feature detection provides the possibility to give additional functionality to modern browsers and instant support for devices that add the support for the feature in the lifetime of the application. In the conference application, I used the Modernizr feature detection library[11] to check for HTML5 features.

For example, the user could add sessions to his or her favorites by clicking

[11] `http://www.modernizr.com/`

the star in the agenda (see Figure 6.4) or on the session details view. The favorites were then listed on the home view (see Figure 6.5) together with information about the time left for them to begin (see requirements 3 and 6 in Section 5.1).

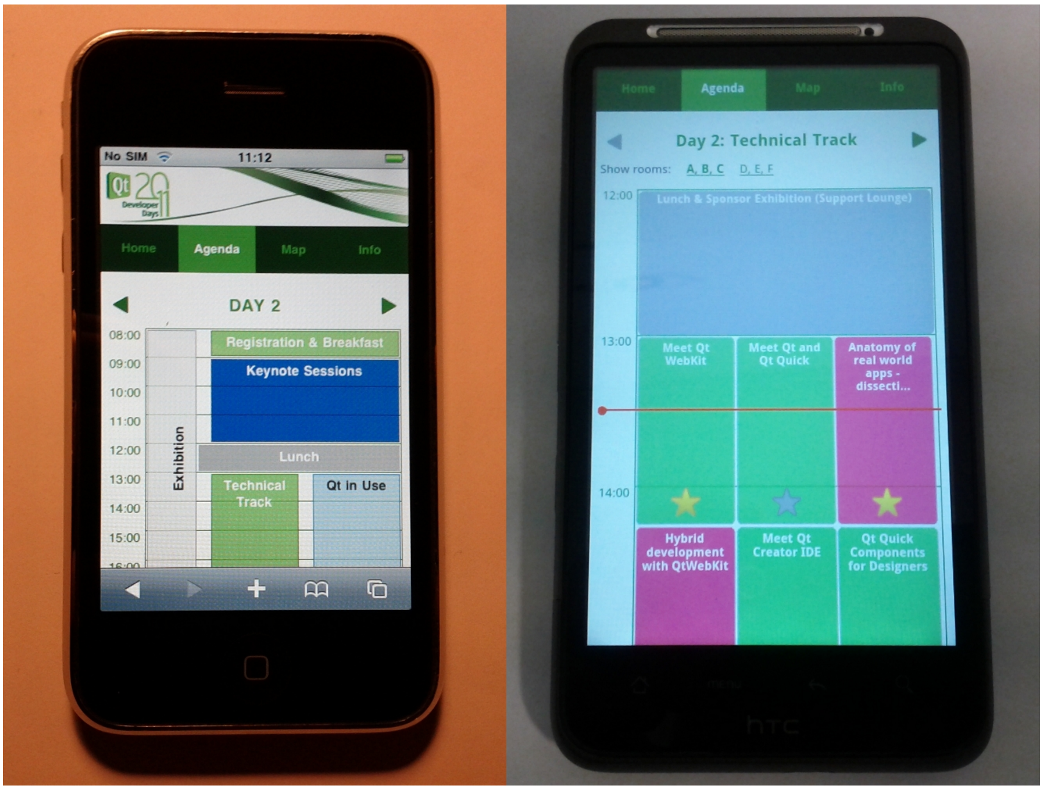

Figure 6.4: Agenda view with stars to add favorites on an iPhone (left) and on a device running Android (right).

I used HTML5 localStorage for storing the favorites in the user's web browser. By using Modernizr, I 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.

6.4 Targeting Different Screens

Probably the biggest difference in various devices and form factors is the screen and its size, resolution, and dimensions. Web applications should ad-

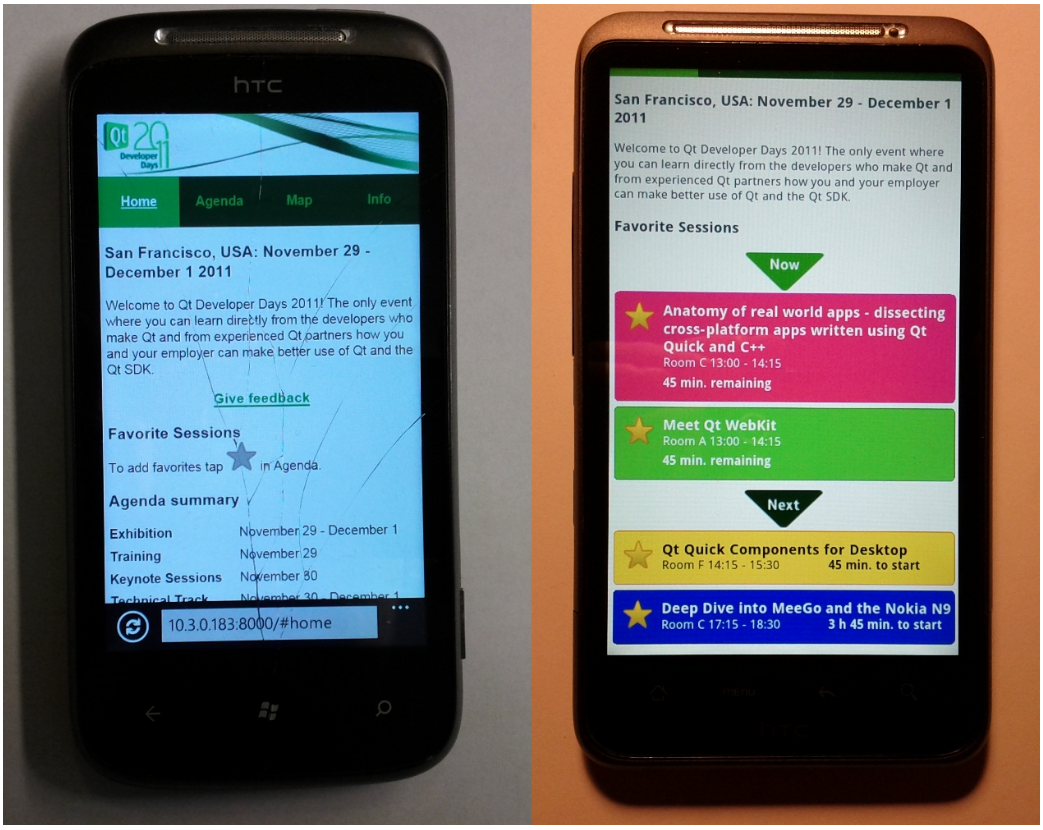

Figure 6.5: Home view without favorites (left) on a device running Windows Phone 7 and with favorites (right) on a device running Android.

just to the available space and flexibly handle screen orientation and window size changes (see requirement 2 in Section 5.1).

First, to target mobile and tablet platforms, the viewport meta information 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 6.3. iOS devices also support these same properties.

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 6.3: Viewport meta tag configuration for Android according to `http://developer.android.com/guide/webapps/targeting.html`

If I 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[12], 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 I used for the conference application (as defined above), I 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 I 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, I used media queries (see Section 3.1) to use better background images for high-resolution screens. I also dynamically set the map view (see Figure 6.6) images based on the screen dimensions so that I could provide smaller images for smaller screens and high resolution images for tablets and other devices with larger screen estate (see requirement 5 in Section 5.1).

6.5 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

[12]`https://developer.apple.com/library/safari/documentation/appleapplications/reference/SafariHTMLRef/Articles/MetaTags.html`

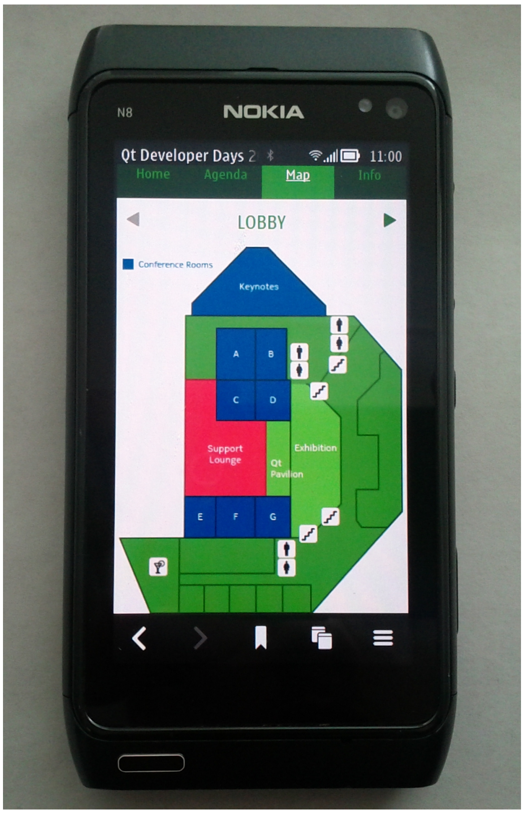

Figure 6.6: Floor map view on a device running Nokia Belle (previously Symbian Belle).

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 at any point switch between the two (see requirement 2 in Section 5.1).

In the conference application, I wanted to have different header and footer background images for different orientations. I 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. The application listened to this event, inferred the orientation from the screen dimensions, and executed the wanted functionality for the event. I also had to do a fallback for Mobile Internet Explorer browser to listen to the window resize event because the browser does not support the orientation change event.

6.6 Handling Mobile Networks

One of the biggest problems in mobile web applications is the network that is often slow and unreliable [65]. 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 (see requirement 4 in Section 5.1). 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.

6.6.1 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, I made sure the data was minimized and compressed with Gzip. Second, using JSON instead of XML in Ajax requests saves bandwidth [9] and needed effort from 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 I could store the application state locally to the browser avoiding the network completely.

6.6.2 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 storage APIs (see Section 2.7.2), 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, I put the most focus on the HTTP caching. Following the performance guidelines specified in Section 4.7, I created unique URLs for all different versions of all static resources (images, CSS files, JavaScript files, and the AppCache manifest file) and set a far future expires header to them. This way I 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 to tell 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 (see requirement 4 in Section 5.1).

The effect of this was clearly seen in the application log files, where the manifest file was first downloaded with the referenced other files on the first page load, but on subsequent page loads only the manifest file was requested for changes.

Client side caching was used in saving the user specific state in the conference application and experimented with the JSONCache library specified in Section 6.2. Using localStorage, I could 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 requested again, the local cache is checked first, and networking can be avoided altogether (see requirement 7 in Section 5.2).

6.6.3 Preloading

One way to prevent user interface slowness due to flaky networks is to preload resources and data that is expected to be used later on. In the conference application I 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.

6.6.4 Offline Support

Using HTML5 AppCache offline manifest file and storing application state in localStorage, I provided full offline compatibility for the conference application. With the offline manifest, I specified the needed resources for all device types as categorized by the rules defined in Section 6.3.1. 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 ended up having indeed a 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, I 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.

I had some problems in developing the application when using the offline manifest. In development mode, I want changes to be shown immediately, and I had to conditionally use the offline manifest based on the environment.

Using the offline manifest poses several additional problems. First, the whole cache is made invalid if only a single resource returns a 404 Not Found HTTP status code. Second, if there are updates in the cache, the updates are indicated in the JavaScript events, but the updated resources are available only in the second page load after the updates. Therefore, updates do not show up immediately when the application is refreshed, and I had to show an additional confirm dialog for informing the user about updates so that they could refresh the page again. This has a somewhat noticeable impact on the application user experience, but was seen as a needed compromise to get the latest conference schedule data and interface updates.

6.6.5 Handling Interruptions

Small interruptions are common in mobile networks [65]. 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 6.2 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 (see requirement 8 in Section 5.2). 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.

6.7 Animations

Animation and transitions, if not overused, can be a valuable addition to the user experience 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 (see Section 3.1). However, the performance of the animations is not yet good enough for a cross-platform mobile application. I 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 4.4) I could have provided enhanced experience for the platforms that support animations well, but only iOS devices performed well enough, so for simplicity I did not use any animations.

6.8 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 Mobile Boilerplate[13] that I used as a base for the conference application. Here I present some techniques and tools that help to improve application performance and reduce bugs.

6.8.1 JSLint

JSLint[14] is a code quality tool for JavaScript. There are several JavaScript features that are suboptimal for performance or code maintainability [11]. 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.

I had automatic JSLint checking integrated in the GNU Emacs[15] editor that I used for all JavaScript programming, which helped to notice common errors as early as possible and made the code cleaner.

6.8.2 Lazy initialization

Postponing work as long as possible is a valuable optimization technique. In lazy initialization, initialization work is minimized in application startup to render the initial view fast. Additional views are then initialized only when they are requested.

[13] `http://html5boilerplate.com/mobile`
[14] `http://jslint.com/`
[15] `http://www.gnu.org/software/emacs/`

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 I tried to postpone all work to be done as late as possible and doing as little work as possible for faster execution.

6.8.3 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. [64]

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 user interface artifacts and slowness. [64]

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 it after the modifications are done. [64]

I used these techniques in the conference application to minimize user interface reflows to improve the perceived performance.

6.8.4 Efficient Event Handling

In an interactive web application, there are 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 [64]. In event delegation only one event listener is attached to a parent element of the elements that we want to observe. Then, in the parent event handler we check the target element of the event and execute the wanted functionality based on the target.

One other optimization for touch screens 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 click event is fired[16]. 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 user interface components a lot more responsive.

I 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. I also used touch events in the session rating form seen in Figure 6.7.

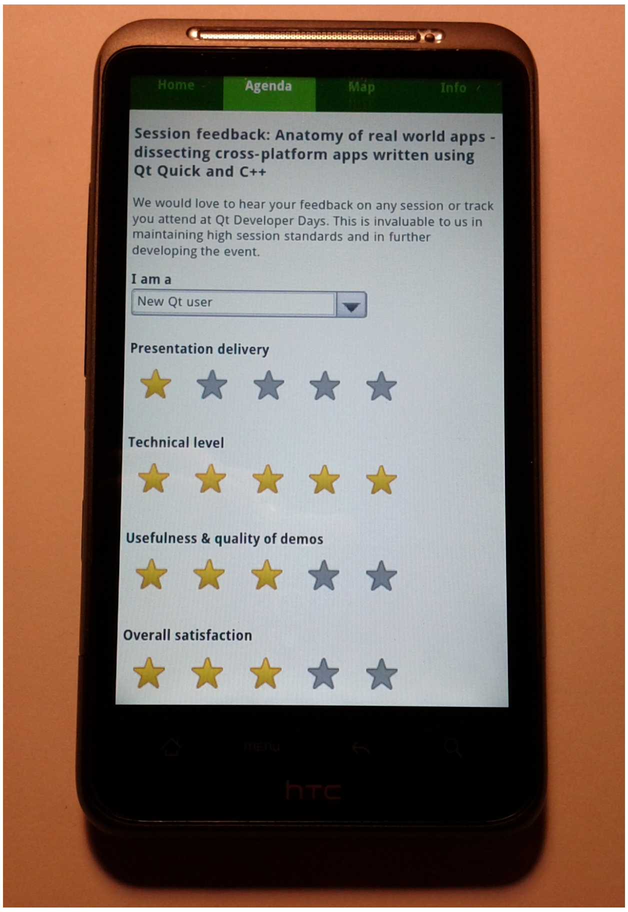

Figure 6.7: Touchable session rating stars on a device running Android.

[16]`http://code.google.com/mobile/articles/fast_buttons.html`

6.9 Performance Analysis

I made a quantitative analysis of the conference application performance by using two different tools: YSlow and Page Speed. These tools analyze the 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 [51, 52] and specified in Section 4.7.

6.9.1 YSlow

YSlow is a website analyzer originally developed by Steve Souders. It checks the website against the rules defined in Section 4.7. I analyzed the conference application performance using YSlow. The results of the analysis are seen in Figure 6.8 and Figure 6.9.

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 (Content Delivery Network) 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 of 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 HTML templates for the single-page application JavaScript code. The `type` attribute of the tags was `text/x-handlebars-template`, which complies with the generic extension mechanism as specified in the HTML5 specification draft [23]. Thus this was not seen as a problem.

6.9.2 Page Speed

Page Speed[17] is an open-source project by Google for analyzing and optimizing web site performance best practices. I used the Google Chrome browser extension to analyze the conference application against the performance rules defined in Page Speed. The results are shown in Figure 6.10.

I was 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 4.7, but there are also additional rules.

The only real problem in the score was the 'Optimize Images' rule. I had not optimized the images used in the application, but instead used the

[17] `http://code.google.com/speed/page-speed/`

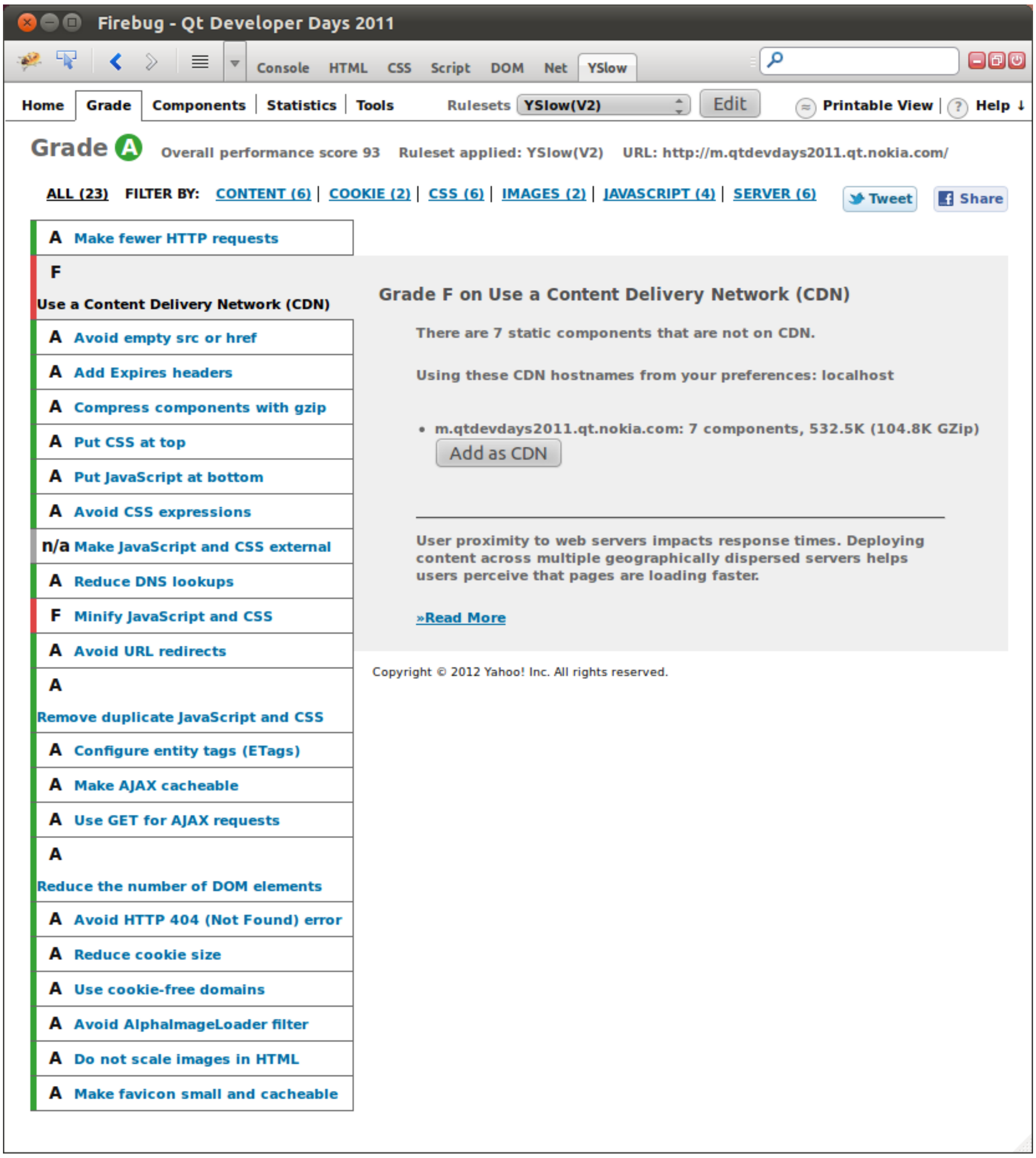

Figure 6.8: YSlow results grade.

images provided by the designers. Going further, I could have saved a lot of bandwidth by optimizing the images with tools such as Pngcrush[18].

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

[18] http://pmt.sourceforge.net/pngcrush/

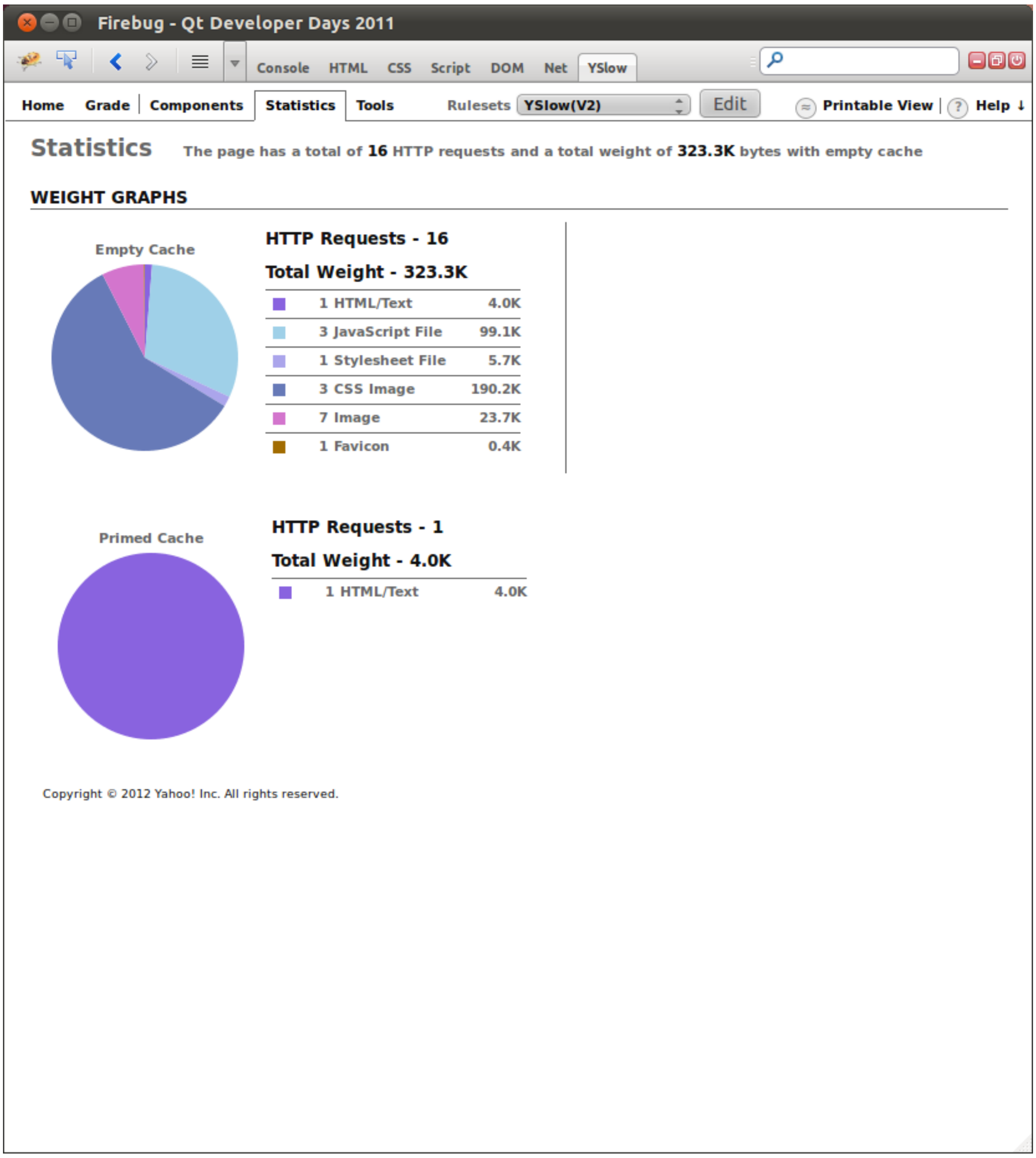

Figure 6.9: YSlow results statistics.

in Section 4.7. However, since I 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 tem-

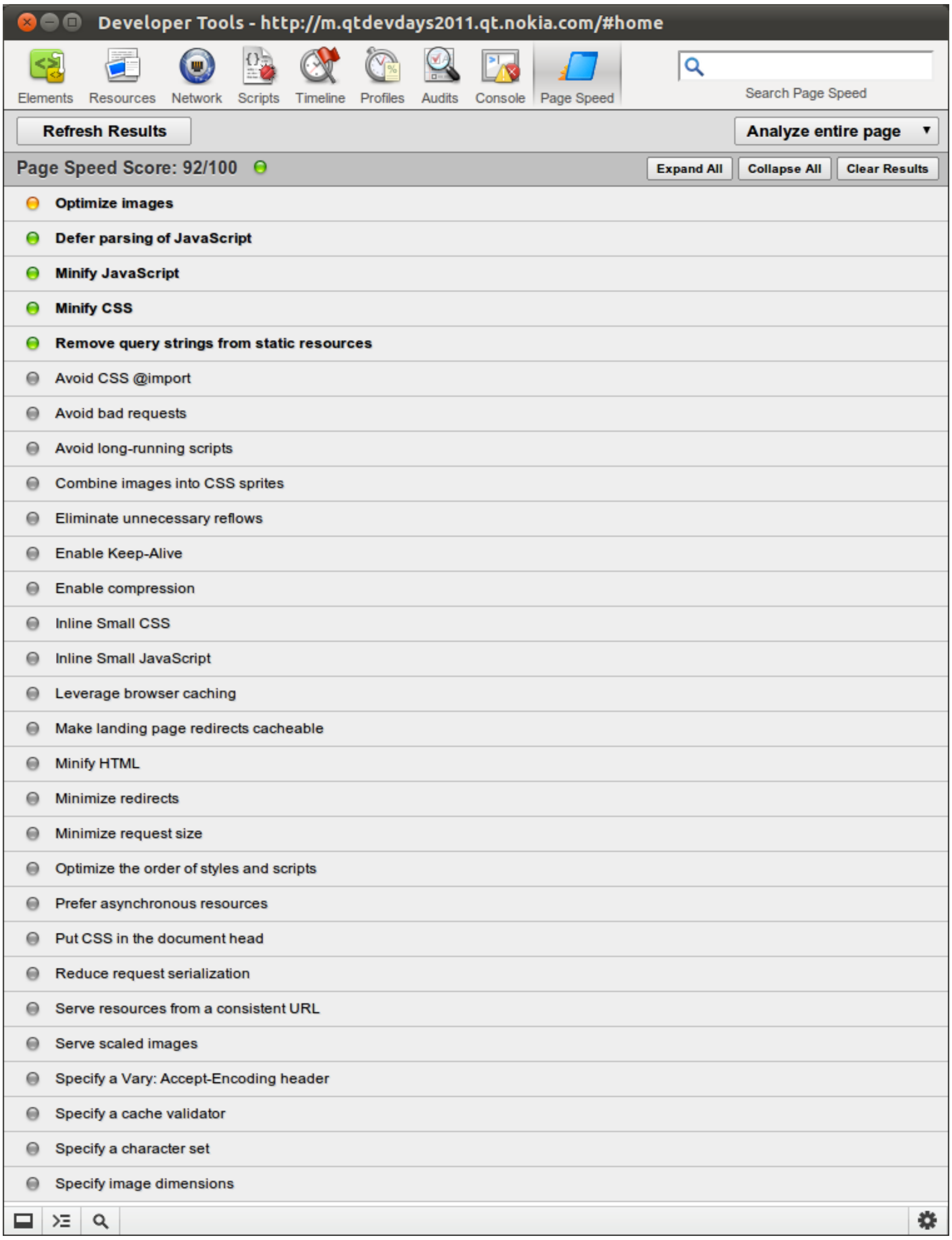

Figure 6.10: Page Speed results for the conference application.

plating library not being minified. All the JavaScript libraries were included in their minified form, but Handlebars library was only available unminified. I also did not want to minify it myself to avoid breaking any functionality. All other JavaScript files were minified and combined 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 delivered Gzipped. 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. I did not change this because the query strings in the static assets were an essential part of our caching strategy.

6.10 Summary of Results

In this chapter, I presented techniques and tools to handle problematic areas in mobile web development. I used several APIs defined in HTML5 and related specifications to tackle these practical challenges.

First challenge was to handle the varying screens and device form-factors (see requirements 1 and 2 in Section 5.1). Feature detection with media queries (see Section 3.1) and the Modernizr library proved to be a very practical and working solution for targeting styles for different device types. Custom `meta` tags in the document as well as orientation handling events were invaluable in setting the viewport dimensions and handling changes in it. However, screen dimension and orientation handling have lots of differences between mobile browsers, and due to the differing APIs and plain browser bugs, handling varying screens and form-factors needed a lot of testing to get the application layout work properly.

Handling mobile networks is a huge challenge in building mobile web applications (see requirement 4 in Section 5.1). I presented many solutions such as minimizing data transfer, caching on different application layers, preloading resources, and adding offline support with HTML5 APIs. These techniques need a lot of know-how and experience, and might not be trivial to implement.

The offline capabilities defined in HTML5 specifications were very valuable and adding offline support ended up being rather simple for a single-page application. However, handling interruptions and unreliable networks is more complex, which is why I developed the JSONCache utility library for caching and fetching data multiple times in unreliable networks (see requirements 7 and 8 in Section 5.2). In my testing, the library worked very well, and using this approach can have a big effect on the end-user experience.

Defining cross-browser stylesheets for different mobile and tablet devices is hard. The CSS support varies between platforms and platform versions, and usually the group of target devices for testing ends up being very large. Animations were a huge pain point in CSS. Adding an animation is very simple using the CSS3 animation rules, but the performance in mobile browsers is very poor to say the least. This is why I dropped all animations from the page changes in the conference application. I hope this is one area where future platform and mobile browser updates put focus on, since animations are an essential part of the user experience of an application.

I provided many techniques for efficient scripting in mobile web applications. In my view, the architecture as well as the DOM and event handling are the most important areas for performance optimization. I provided solutions like lazy initialization, document fragments, and event delegation as well as introduced the JSLint tool for automatically checking code quality.

Last, I did a quantitative web performance best practices analysis on the conference application. Using the YSlow and Page Speed tools, I got very high scores and explained the areas for further improvement. These automatic tools are valuable in optimizing the overall performance, since they point out the most problematic areas for improvement, and many of the solutions can be very simple, such as configuring the web server without even touching the application itself.

To answer the research questions posed in Section 1.1, I sum up as follows:

- **RQ1**: *What are the main problem areas in mobile web development?*

  The main problems are targeting different types and sizes of screens and resolutions, handling the unreliable networks and offline modes, and user interface performance, especially with animations.

- **RQ2**: *Do HTML5 and related specifications solve these problems?*

  The specifications offer a good way to target different screens with media queries, but browsers have a lot of differences that have to be tested for.

  Offline support is relatively easy to provide, but handling interruptions and unreliability have to be tailored in the application architecture.

  Performance optimization is the job of the application developer, and animation performance is hard to optimize for, leaving future updates for the browsers as the only viable option.

- **RQ3**: *What other practical means do we have to solve these problems?*

Custom meta tags are very useful for configuring the viewport and sound architectural choices and best practices help optimizing the application performance.

Chapter 7

Conclusions

In this chapter I present the further improvements for our implementations and the final conclusions of this work.

7.1 Further Work

At the time of implementing the conference application, I used the presented tools, YSlow and Page Speed, for analyzing the performance best practices of the application. However, there are other and newer tools especially targeted for analyzing mobile web application performance practices. For example, the online version of Page Speed offers rules for mobile performance optimization[1], which add some mobile-specific optimizations. This service would probably offer a better alternative for mobile web application optimization than the versions I used in the performance analysis.

One of the major optimizations I was aware of, but did not do, was optimizing images. This was a rather large area where I could have had significant improvements in the application performance. Going further, combining several images into one large image sprite would reduce the number of HTTP requests, and using lossless or lossy image optimization tools, the size of the images could be further reduced.

The conference application targeted the high-end smartphones that the users were expected to have in a developer conference. However, a typical mobile web application does not have this advantage in its target users. Using progressive enhancement techniques, we can start from the lowest performing devices and build from there up to the latest and best-performing devices. This also goes with the very idea of the Web by providing a truly open and universal access to the applications.

[1] `http://code.google.com/speed/page-speed/docs/mobile.html`

7.2 Discussion

The Web revolutionized the way we communicate, consume and produce information in ways that could not have been foreseen twenty years ago. In addition, the mobile revolution has spread the Web from our home desks to anywhere we are, to be used at any time of the day. The roots of the Web lie in openness and universal accessibility for everyone, and today more and more people can afford a device to access the vast information spread all over the Web.

One crucial factor in the universality is the open standards used for defining the protocols and APIs of the Web. HTML5 tackles many of the growing pains of the Web by defining standards to handle all the devices capable of accessing the Internet. The set of new specifications or specification drafts is very large, and growing all the time.

In this work, I introduced the latest specifications and drafts related to modern web application development. Some of these specifications already have very good implementations in several browsers, but some are just very early drafts. I also presented modern tools and libraries for developing mobile web applications.

Performance is one of the main components of a successful and usable application. In this work, I took a practical focus on performance optimization of mobile web applications. I also tackled other problem ares in developing these HTML5 applications.

I implemented a schedule application for a developer conference and a utility library for handling unreliable mobile networks. The conference application was successfully used in two conferences by hundreds of people, and the received feedback was excellent. I had to solve a lot of problems and research solutions in areas that were new to me. I used the latest HTML5 and related APIs in several parts of the implementation.

The Web is living very interesting times, and universality in geography and device types is growing. The browser is the culmination point of all the new development of the Web, and the new open standards make the browser a powerful platform for a vast array of different applications.

Keeping the Web open and accessible for everyone is the key for technological advancement and innovation in the future. The Web is here to stay, and with the potential of HTML5 and other modern tools, we can build powerful applications that improve our day-to-day lives, as well as applications that revolutionize our lives. There is grandeur in this view, but without idealism and relentless pursue of universal accessibility, the full potential of the Open Web might never be reached.

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] BLOCK, S., AND POPESCU, A. DeviceOrientation Event Specification. W3C Editor's Draft, W3C, Jul 2011. Available at: http://dev.w3.org/geo/api/spec-source-orientation.html. Accessed 12-Febuary-2012.

[6] 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.

[7] 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.

[8] 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.

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

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

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

[12] 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.

[13] 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.

[14] Fielding, R. T. *Architectural Styles and the Design of Network-Based Software Architectures*. PhD thesis, University of California, 2000.

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

[16] Fling, B. Anatomy of a HTML5 Mobile App. Available at: `http://pinchzoom.com/posts/anatomy-of-a-html5-mobile-app/`. Accessed 22-Febuary-2012.

[17] Fraser, S., Jackson, D., Hyatt, D., Marrin, C., O'Connor, E., and Schulze, D. CSS Transforms. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dev.w3.org/csswg/css3-transforms/`. Accessed 12-Febuary-2012.

[18] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1995.

[19] 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.

[20] Graham, S., and Mielczarek, T. Gamepad. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dvcs.w3.org/hg/gamepad/raw-file/default/gamepad.html`. Accessed 22-Febuary-2012.

[21] Gregg, J. Web Notifications. W3C Editor's Draft, W3C, Sep 2011. Available at: `http://dev.w3.org/2006/webapi/WebNotifications/publish/Notifications.html`. Accessed 22-Febuary-2012.

[22] 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.

[23] 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.

[24] 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.

[25] Hickson, I. HTML5 Web Messaging. W3C Editor's Draft, W3C, Jan 2012. Available at: `http://dev.w3.org/html5/postmsg/`. Accessed 12-Febuary-2012.

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

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

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

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

[30] Jackson, D., Hyatt, D., Marrin, C., and Baron, L. D. CSS Transitions. W3C Editor's Draft, W3C, Jan 2012. Available at: `http://dev.w3.org/csswg/css3-transitions/`. Accessed 12-Febuary-2012.

[31] Jackson, D., Hyatt, D., Marrin, C., Galineau, S., and Baron, L. D. CSS Animations. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dev.w3.org/csswg/css3-animations/`. Accessed 12-Febuary-2012.

[32] Kostiainen, A. Vibration API. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dev.w3.org/2009/dap/vibration/`. Accessed 22-Febuary-2012.

[33] Kostiainen, A., and Lamouri, M. Battery Status API. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dvcs.w3.org/hg/dap/raw-file/tip/battery/Overview.html`. Accessed 22-Febuary-2012.

[34] Laine, M., Shestakov, D., Litvinova, E., and Vuorimaa, P. Towards unified web application development experience. *IT Professional*, 99 (2011), 1–1.

[35] Lamouri, M. The Network Information API. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dvcs.w3.org/hg/dap/raw-file/tip/network-api/index.html`. Accessed 22-Febuary-2012.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

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

[42] 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.

[43] 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).

[44] 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.

[45] Parker, T., Toland, P., Jehl, S., and Wachs, M. C. *Designing with Progressive Enhancement: Building the Web That Works for Everyone*. New Riders Publishing, 2010.

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

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

[48] 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.

[49] 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.

[50] Scheib, V. Ppointer Lock. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dvcs.w3.org/hg/pointerlock/raw-file/default/index.html`. Accessed 22-Febuary-2012.

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

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

[53] Steen, H. R. M. Clipboard API and events. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://dev.w3.org/2006/webapi/clipops/clipops.html`. Accessed 22-Febuary-2012.

[54] 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.

[55] 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.

[56] Tibbett, R. Contacts API. W3C Editor's Draft, W3C, Feb 2012. Available at: `http://w3c-test.org/dap/contacts/`. Accessed 22-Febuary-2012.

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] Van Kesteren, A., and Çelik, T. Fullscreen. Editor's Draft, Feb 2012. Available at: `http://dvcs.w3.org/hg/fullscreen/raw-file/tip/Overview.html`. Accessed 22-Febuary-2012.

[62] 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.

[63] 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.

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

[65] Zandy, V. C., and Miller, B. P. Reliable Network Connections. In *Proceedings of the 8th annual International Conference on Mobile Computing and Networking* (2002), ACM, pp. 95–106.

[66] Çelik, T., Etemad, E. J., Glazman, D., Hickson, I., Linss, P., and Williams, J. Selectors Level 3. W3C Recommendation, W3C, Sep 2011. Available at: `http://www.w3.org/TR/selectors/`. Accessed 12-Febuary-2012.