GRADE

Master's Thesis

LUVIT for mobile platforms

By

Anders Forslund and Christian Veinfors

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

Abstract

Luvit is a web based learning management system used by about one hundred companies, authorities, organizations and universities in seven different countries. The purpose of this project was to examine and develop a solution that worked well on mobile devices such as mobile phones and tablets.

The first part of this thesis investigates what kind of solution is best suited for Luvit. A mobile web based one that works on several platforms or an application that is developed in the native language of the device. The main reason for choosing a web solution is that it is easy to maintain since there is only one single application. The main advantages of native solutions are their superior performance and that they have access to more hardware functionality.

Different web based solutions were tested and compared against each other. With the help of developing prototypes the choice was made to develop Luvit Mobile using the JavaScript frameworks Sencha Touch and PhoneGap.

In the second part of the thesis the application was developed using these frameworks. Sencha Touch was used to build the application so that it looked like a native one. PhoneGap was used to wrap the application and compile/deploy it on different platforms such as Android and iOS. This makes it feel like a real application and not a web based one.

Acknowledgments

This Master’s thesis would not exist without the support and guidance of Rickard Nygren, CEO at Grade, and our supervisor Christian Nyberg, associate professor at LTH. We would also like to thank Stefan Larsson and Anna Johansson at Grade for their help.

Table of Contents

CHAPTER 1

1 Introduction

In 1997 a learning management system called Luvit [1] was created. It is today used by approximately one hundred companies, authorities, organizations and universities in seven different countries. Luvit was a result of a project initiated by system developers and teachers at Lund University because they needed a tool for distance learning [3]. In 1998 the company Luvit AB was started.

Three years earlier, in 1995, the company Grade was started [2]. They worked mainly with a course development tool called Composer and a small learning management system called Maestro.

Luvit AB and Grade merged in 2008 and became just Grade. They scrapped Maestro and decided to concentrate only on the Luvit Learning Management System (LMS). Grade is today a subsidiary of the company Avensia Innovation and has offices in Lund and Stockholm.

There are several competing LMS to Luvit, for instance *PingPong*, *E-gate* and *It's Learning*. The latter one is the only one that offers mobile solution. It has limited functionality and lacks more advanced features.

The use of modern mobile devices, such as smart phones and tablet computers has increased significantly in the recent years. Consequently, the demand for mobile friendly web applications has grown larger. With bigger displays, better hardware and more sophisticated software, the potential for web solutions is better than ever before.

Because of this, there is now a suitable opportunity for Luvit to take the step into the mobile world. Regular Luvit is not very user-friendly on small devices, mainly because of the huge amount of functionality and information that doesn't work well on small screens. The availability of the

learning system would increase a lot with a mobile version since many people today have access to handheld devices with an Internet-connection.

This master's thesis consists of two parts. In the first, different tools and frameworks will be examined and compared with each other to find the most appropriate solution according to performance, functionality, documentation and platform support. In the first part (chapter 3-4), different web based solutions will be examined to see if they measure up with native solutions. The second part (chapter 5-6) will be to develop the actual application based on the results in the first part. The development methodology that will be used for this project is inspired by *eXtreme Programming*[1] *(XP)*. Requirements will be added as the project progresses so the development has to be carried out iteratively. Also, pair programming will be used when more difficult tasks are developed.

[1] *eXtreme Programming* is a methodology that has been used by the authors of this thesis during the education at Lunds Tekniska Högskola in different courses.

CHAPTER 2

2 About Grade and Luvit

2.1 *Parts of Luvit*

Luvit consists of three different parts; Luvit Portal, Luvit Education and Luvit Administration.

Portal is the first view the user sees after logging in to the system. An overview is shown of the different courses and study programs the user is participating in. It is also possible to communicate with other course participants, view the calendar and read the latest news.

Education is where the actual course takes place. Here, the course material is presented, users can communicate/collaborate with each other, assignments can be handed in and much more. This is the part in which a typical user spends most of the time in Luvit and actually does most of the work.

Administration is an administrative tool for managing courses and users in Luvit. Here, it is also possible to manage which menus and functions that are shown in Education and Portal.

In this thesis, the authors will only concentrate on Luvit Education.

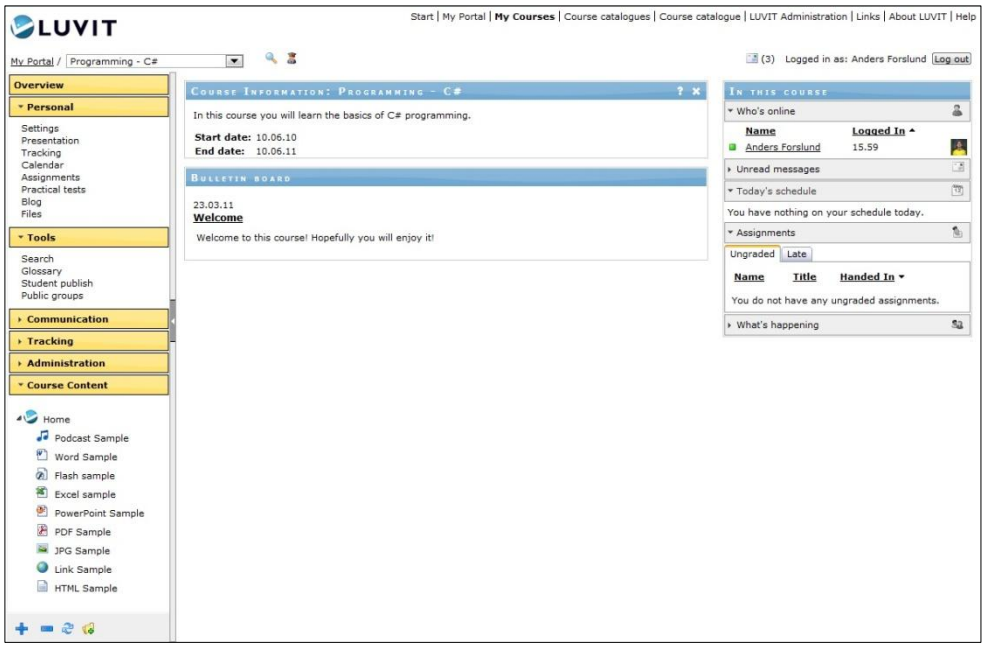

Figure 1: Start screen in an example course in Luvit Education

2.2 *Under the hood*

Luvit is a web-based system. It was from the beginning built in Active Server Pages with COM-components developed in Visual Basic. In 2005 the system was entirely re-designed.

Today, the front-end interface is built with HTML, CSS and JavaScript. This means that it works on all platforms that have a modern web browser, like Windows, Linux and Mac systems.

The back-end is built with Microsoft technology. ASP.NET is used with the language C# which communicates with a database built in Microsoft SQL Server.

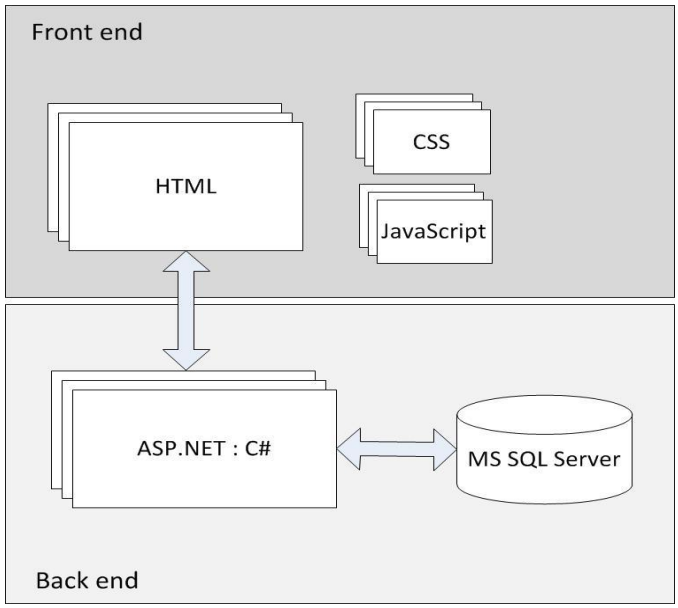

Figure 2: Software design of Luvit

CHAPTER 3

3 Introduction of the frameworks

3.1 *Why web based mobile applications*

There are two ways of building a mobile application. One approach is to develop in the native language of the device. The other way is to make a web based solution that works for different devices and platforms [4].

A mobile web application is not installed on the device but is instead reached from a web browser as opposed to native applications. Native applications also have access to a phone's hardware and core functionality, like GPS, camera and vibrator.

There are many advantages of web applications. One is that the developers only have to maintain and develop *one* application that works for several platforms. The availability of the application is better because it is directly reachable from any device that has a web browser. Also, as a web developer it is easier to get into the development process in contrast to learning a whole new programming language.

One of the biggest drawbacks of a web-based solution is the degraded performance when using JavaScript frameworks. The hardware of a mobile device has limited performance and the application might not execute as smoothly as desired since JavaScript can be quite demanding.

One advantage when developing native applications is, as mentioned earlier, the easy access to core functionality of the mobile device. You have more freedom to make use of the entire phone, and not just the web browser. Another big benefit is that the application often runs more smoothly, because it can utilize the hardware better.

To make a mobile web based application look and feel native, there is a wide range of different frameworks and tools to choose from. As the mobile

market has increased during the last years, several new frameworks have emerged. Since it is not possible to examine all frameworks that exist, the most popular ones will be selected. The results can be seen in the following chapters.

3.2 *Sencha Touch*

Sencha Touch is, according to their homepage, the *first* HTML5 based mobile web application framework [5]. It offers developers access to a variety of components like buttons, lists and icons. The framework also has support for animations, touch events and local data storage. As mentioned above, it takes advantage of HTML5 but also CSS3 and JavaScript.

The framework generates its own DOM[1] (Document Object Model) instead of enhancing existing HTML code. This means that the HTML is created dynamically, either manually by the developer or automatically by Sencha Touch. This makes it feel like developing a real application instead of a web based one.

Sencha has earlier offered several tools using JavaScript for production of feature rich web sites. Sencha Touch was created to build rich web applications for mobile devices. They officially support iOS, Android and BlackBerry but works with all WebKit[2] based browsers, like Google Chrome and Safari.

Sencha Touch has support for touch events, like pinch and rotate gestures. It also has automatic adaption to screen resolutions which is an important feature since resolutions differ much between devices.

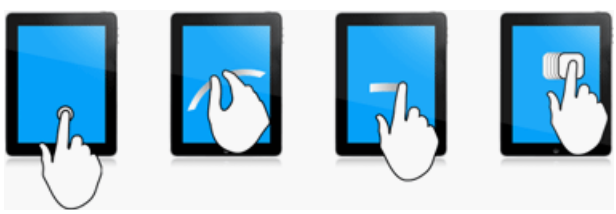

Figure 3: Examples of touch events supported by Sencha Touch [Online image] http://www.sencha.com/products/touch/

[1] The DOM is an interface that scripts can use to access and dynamically update the content and structure of a document, often a HTML document [18]
[2] WebKit is a layout engine that helps web browsers to render web pages correctly [19]

The theming in Sencha Touch is made in CSS3 with the help of SASS[1]. This makes it relatively easy to change the appearance of the application for developers familiar with regular web development. CSS3 can, in contrast to its predecessors, make use of rounded corners and gradient backgrounds. It can also be used for animations and Sencha Touch utilizes this in their product allowing flexible and easy animations between screens and views.

The mobile framework uses HTML5 for video and audio playback but also for offline data storage, making it possible to develop applications that can function offline as well.

3.3 *jQuery Mobile*

The main focus of jQuery Mobile, which is an open source project, is to make it possible for developers to "write less, do more" [6]. What this means is simply that development of *one* web application should be enough to make it available for the most common mobile platforms.

jQuery is a well-known JavaScript library that has been extensively used for web development during the past years. jQuery Mobile was announced in august 2010 and is at the time of writing only available in an alpha version.

This JavaScript framework provides developers with a library of different user interface components such as lists, buttons and form elements. The framework is built using standard semantic HTML aiming at making the applications accessible from a wide range of devices, for example iPhone, Android, BlackBerry, Windows Phone, Palm and Symbian.

Unlike Sencha Touch, the developer has to manually write the HTML code making it more like traditional web development.

The library is small in size, around half the size of Sencha Touch when it is minified[2]. This makes it fast to download even on devices with poor internet access.

[1] SASS is a meta-language that improves the functionality of CSS and adds features like selector inheritance and support for variables [20].
[2] Minified is a term describing that a library or similar is reduced in size by removing white space characters and comments for instance [21].

jQuery Mobile has support for basic touch events like tap and swipe but lacks support for more advanced touch gestures like pinch and rotate.

3.4 *Titanium Mobile*

Titanium Mobile differs from the previously mentioned tools. The code written for the application is still in JavaScript but is translated to the native language of the desired device [7]. This makes it look and feel even more like a real native application and offers potential for better performance. The appearance follows the standard look of the device. This means that an application developed in Titanium Mobile for a specific platform will look like a regular application developed in the native language.

Titanium Mobile supports iOS and Android. An application must be adjusted to fit a specific platform in order to access different features of the device.

Applications created with this tool cannot be accessed from a web browser in the same way as applications developed with Sencha Touch or jQuery Mobile since the source code is compiled to the native language of the device. Therefore it has to be run as a regular application. Thus, applications created in this tool are not considered to be web applications even though they are written in JavaScript.

One of the benefits with Titanium is that core functions of a device can be used which enriches the applications.

3.5 *PhoneGap*

In April 2009, PhoneGap won the Web 2.0 Expo LaunchPad competition. Since then, it has been downloaded over 350,000 times [8].

Unlike the other frameworks in this chapter, PhoneGap cannot be used by itself. The actual user interface needs to be built by the developer. The idea of this is that the developer can use whatever web technology best suitable for the situation. This means that PhoneGap can be used together with both jQuery Mobile and Sencha Touch in a good way.

PhoneGap is actually a solution that wraps the web application and turns it into a real application that is installed on the device instead of running on a web browser. Because of this, native features of the device can be accessed.

It has support for all the larger mobile platforms available today, like BlackBerry and Symbian but most of the features work best in Android and iOS devices.

PhoneGap provides a cloud[1] based build service which allows the developer to upload the web application and get back the executable application for the chosen platforms. All the actual compilation gets done in the cloud.

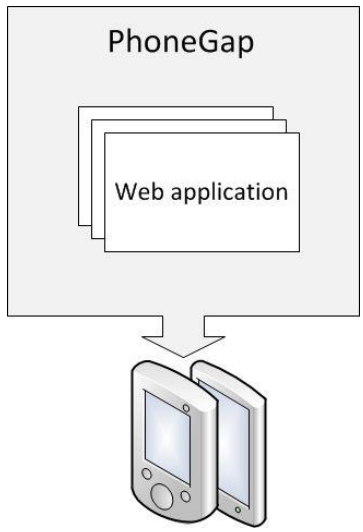

Figure 4: the wrapping of PhoneGap

[1] The definition of Cloud Computing is, according to Sveriges IT-arkitekter, "applications that are delivered as services over the Internet but also the hardware and software that provide these services" [22].

CHAPTER 4

4 Comparison of the frameworks

The purpose of the initial part of this project is to investigate and compare possible tools for the development of Luvit for mobile platforms. The chosen tool must meet several requirements in terms of performance, platform support and interface appearance. It is important that the tool provides good usability for the end user and also that it is relatively easy to use for development. For the assessment of these criteria, prototypes in the different frameworks are being developed.

To be able to make a fair evaluation it is important to make the prototypes as alike as possible, both regarding appearance and functionality. The prototypes are also tested on the same devices. These are Apple iPhone 3G with iOS 4.1, Apple iPad with iOS 4.3 and ZTE Blade with Android 3.3.

4.1 *Interchange of data*

The final solution of Luvit Mobile needs to be able to both read and write to the database of Luvit. Therefore, it is a good idea to examine possibilities for this in the prototypes instead of just presenting "dummy data". One might think that it is just as simple as re-using existing code from Luvit for the prototypes but it is more difficult than that since different technologies are going to be used.

Usually in an ASP.NET project, every page has its own code-behind file. The page-files, with the extension ".aspx", contain static HTML-markup and markup that define so called ASP.NET Web controls. These controls are tags that are understood by the server, i.e. the code-behind file. An example is shown below with an ASP.NET Label control.

Test.aspx

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "---//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Example page</title>
</head>
<body>
 <form id="form1" runat="server">
   <div>
     <asp:Label runat="server" id="LblHello" />
   </div>
 </form>
</body>
</html>
```

Test.aspx.cs

```
public partial class HelloWorld : System.Web.UI.Page
{
  protected void Page Load(object sender, EventArgs e)
  {
     LblHello.Text = "Hello, world";
  }
}
```

When the page loads, the text property of the label is set to “Hello, world” in the code-behind and is presented to the user in the aspx page.

The code-behind approach is a way to separate content from presentation in contrast to the classic ASP where the server-side code usually was mixed within the HTML-markup and therefore difficult to handle when pages turned large.

This approach is not possible with most of the tools and frameworks in this chapter. Regular HTML-files need to be used and since the examined frameworks and tools use JavaScript that is executed on the client-side it is not possible to call the functions in the code-behind class since that is executed on the server-side *before* the client-side code.

So how does one pass information from a database to a JavaScript framework? The solution is to use a web service or a HTTP handler. In this

case a HTTP handler is used. The handler is a single file with the extension *.ashx*.

Handler.ashx

```
<%@ WebHandler Language="C#" Class="Handler" %>
using System;
using System.Web;


public class Handler : IHttpHandler
{
  public void ProcessRequest(HttpContext context)
  {
     context.Response.ContentType = "text/plain";
     context.Response.Write("Hello World");
  }


  public bool IsReusable
  {
     get
     {
        return false;
     }
  }
}
```

ProcessRequest is the method that will be invoked when the handler is requested and it is here the communication with the database will occur later. In the case above, all it does is returning the text “Hello World” but in a real world application it most probably makes a call to some sort of database and returns that response to the web application. That is exactly what will be the case in the prototypes in this chapter.

The data that is returned is usually in the form of *XML* or *JSON*. XML is the older of the two and its syntax can be seen below.

XML syntax

```
<card>
  <fullname>Anders Forslund</fullname>
  <emailaddrs>
     <address type='work'>anders@work.com</address>
     <address type='home'>anders@home.com</address>
  </emailaddrs>
  <age>25</age>
</card>
```

JSON is a newer format and has gained popularity in the recent years due to its simplicity. For this reason it will be used in the prototypes. The syntax is shown below.

JSON syntax

```
{
  "fullname": "Anders Forslund",
  "emailaddrs": [
     {
        "type": "work",
        "value": "anders@work.com"
     },
     {
        "type": "home",
        "value": "anders@home.com"
     }
  ],
  "age": "25"
}
```

The frameworks have good support for handling both of the formats.

4.2 *Sencha Touch*

The first framework that is going to be reviewed is Sencha Touch. As mentioned before, this framework has been around for a while and thus one can expect great things from this it.

In the moment of writing, version 1.1.0 is the latest one so therefore that is the obvious choice. The Sencha Touch installation folder contains various example applications and a large application that shows off most of the features of the framework. This application is called *Kitchen Sink*. Here, one can try out the functions live and get code examples for them. Also, a small tutorial is provided on how to get started with the development.

4.2.1 Getting started

To try out the Sencha Touch SDK a simple Hello World application will be created. The application should consist of a HTML-file, the Sencha Touch library file and the Sencha Touch standard stylesheet file.

The HTML file should contain references to the files mentioned above.

Index.html

```
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Hello World</title>
    <script src="sencha-touch.js" type="text/javascript"></script>
    <link href="sencha-touch.css" rel="stylesheet" type="text/css" />
  </head>
  <body></body>
</html>
```

As expected, nothing happens when visiting index.html from a browser. Some code needs to be included that utilizes the Sencha Touch library. Usually, this code is included in a separate JavaScript file and referenced in index.html but in this simple case it is placed directly in the HTML file.

Code snippet in index.html

```
<script type="text/javascript">
  new Ext.Application({
    launch: function() {
      new Ext.Panel({
        fullscreen: true,
        html: 'Hello World'
      });
    }
  });
</script>
```

This simple application creates a panel which contains some HTML, in this case the text “Hello World”.

4.2.2 Developing the prototype

The example above is a good way of getting started but the prototype needs to be more advanced than that. It is a good idea to include regular components such as lists and forms which will be suitable in the final product.

First, a general layout needs to be created. A common approach, using Sencha Touch, is to use a card layout. This means that the application consists of different cards, each containing different components. Only one card at a time can be visible.

To begin with, an *Ext.List* is created containing course participants for an arbitrary course. This is our first card. To fill a Sencha Touch list component with information, an *Ext.data.Store* is used. An AJAX[1] call is made to the HTTP handler which returns the users in JSON format and loads it into the store.

Example of filling a store component with data. Irrelevant code is omitted.

```
var usersStore = new Ext.data.Store({
  proxy: {
    type: 'ajax',
    headers: { 'Content-Type': 'application/json;charset=utf-8' },
    dataType: 'json',
    url: 'Handler.ashx?function=GetUsers&courseID=5186'
  }
});
usersStore.load();
```

After this, the store can be binded to the list. To make it look more professional the *Ext.Toolbar* component is added as well. A toolbar can contain a headline, buttons and other components that can help navigation in the application. A useful ability with the toolbar in Sencha Touch is that it is "dockable", meaning that it can be locked into a fixed position and still be seen when scrolling in the application.

To try out some transitions and interactions with the application, more functionality is added. Each item in the list has a listener that reacts to a tap event. When tapping a list element, in this case a course participant, the application switches cards with a slide transition to a second card which will show the user info.

[1] AJAX (shorthand for "asynchronous JavaScript and XML") is a group of techniques used to make web applications more interactive. One example is the ability to make calls to the web server without reloading the page [23]

The switching of cards

```
mainPanel.setActiveItem(cardUserInfo, {type: 'slide'})
```

Before the card switch, another AJAX request to the handler is made to collect user information about the tapped user. This is then loaded into an *Ext.form.FormPanel* which, in this case, is the second card.

The form panel contains two text fields, one which holds the name and one with the e-mail address of the user. To try out some more components, the text fields are put into a *fieldset*. This makes them appear in a group with a nice headline.

The form panel also contains a button, for saving changes made in the e-mail field. This is, once again, taken care of with the HTTP handler that makes the changes to the database.

Another thing worth mentioning is that a back button is added to the toolbar to be able to go back to the previous card. This needs to be done manually and is not taken care of by the framework.

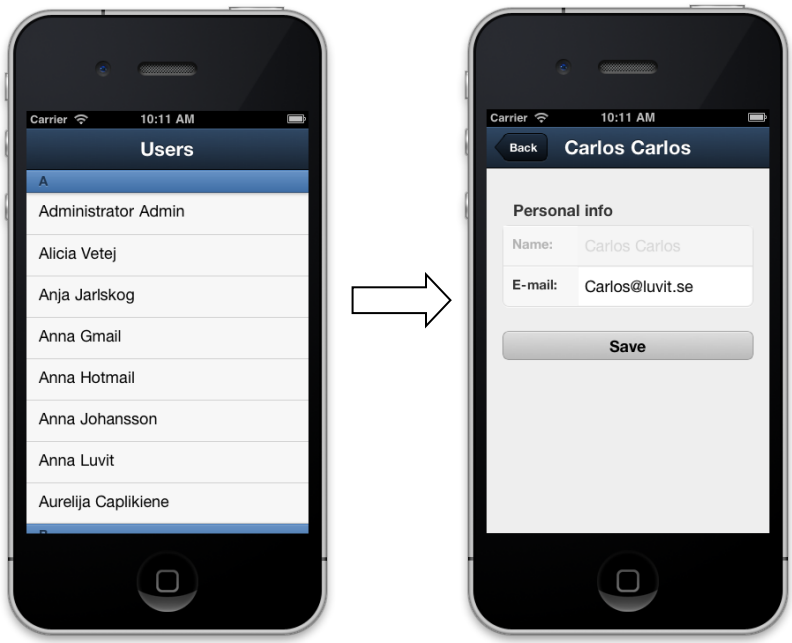

Figure 5: The Sencha Touch prototype

4.2.3 Performance and appearance

Testing the prototype on actual mobile devices is a good way to get an idea on how the framework performs.

As the Sencha Touch library is rather large, it takes some time to load the web application. The loading time is around twice as long as the loading time for the jQuery prototype.

When adding additional functionality to lists and other components, the performance gets reduced on some of the devices. It is therefore important to find a balance between an appealing appearance and performance of the application.

Animations and transitions work better and more smoothly on iOS devices than with Android. This is probably because Android devices do not utilize the GPU when using the web browser [9].

The general user experience with Sencha Touch and the performance is that it is not as smooth as a native application but works very well on newer Apple devices and satisfactorily on newer Android devices. Some

choppiness can occur but not as much that it interferes with the user experience.

The standard appearance of the framework looks a lot like the user interface of an iPhone. The application gets a professional look without much effort at all.

The Sencha Touch installation also comes bundled with other themes. Stylesheets are available to make the web application look like a native BlackBerry, Android or iPhone application.

To create custom themes, SASS is used. This process starts with creating a file with the file extension ".scss". There are two ways of affecting the appearance of the applications. One of them is to override some standard variables that exist. There are about 50 variables available, for instance *$base-color*, *$base-gradient* and *$tabs-bottom-radius.* Most of the variable names are self-explanatory, which makes them easy to use.

Example of variable overriding

```
$base-color: #7A1E08;
$base-gradient: 'glossy';
$body-bg-color: #fbf5e6;
```

The other way of adjusting the appearance is to just use regular CSS.

Example usage of regular CSS

```
body {
  font-family: Georgia;
  color: #5a3d23;
}
```

When satisfied with the adjustments, Compass is used to compile the file into a regular CSS file that can be included in the web application. Compass is a stylesheet authoring tool recommended by the Sencha Touch crew [10].

In Sencha Touch there are also over 300 different icons included that one can use in a web application. The icons are very stylistically pure which make them suitable for most applications.

4.2.4 Functionality

The most commonly used functionality in Sencha Touch is of course the one related to the user interface. The library provides components like panels, buttons, lists, toolbars, tabs, dialogs, form elements and much more. Below is a small example of how a component can be used.

Creation of a tab bar

```
var bar = new Ext.TabBar({
  dock: 'top',
  items: [{text: 'Example button'}]
});

var panel = new Ext.Panel({
  dockedItems: [bar]
});
```

An essential part of Sencha Touch is the support for different touch events. Examples of touch events that can be handled are tap, doubletap, swipe, pinch, touchmove and about 10 more. Worth mentioning is that jQuery Mobile, for instance, only has support for half as many touch events.

Different techniques for fetching data are supported. As mentioned before, AJAX is one of them but JSONP and YQL can also be used.

JSONP, or "JSON with Padding", is used when collecting data from a server in a different domain since this is not possible with AJAX [11].

YQL, or Yahoo Query Language, is an SQL like language that lets the developer query and filter data from a web service [12].

Sencha Touch also supports embedding of audio and video in a simple way.

Embedding of an audio file in a panel

```
var audioPanel = new Ext.Panel({
  items: [{
     xtype: 'audio',
     url: 'test.mp3',
     loop: true
  }]
});
```

4.2.5 Developing in the framework

Even for experienced web developers Sencha Touch can be rather difficult to develop in. When browsing through various forums and mailing-lists on the Internet, people in general have complaints about the steep learning curve.

Sencha Touch works differently from more common JavaScript frameworks like jQuery and Prototype. It creates the interface, i.e. the HTML code, dynamically and it is harder for a developer to have full control of what is happening when accustomed to the more static approach that the latter frameworks use.

The developer does not write any HTML code in the actual HTML file, but has to use either regular JavaScript or the Sencha Touch API.

A typical Sencha Touch application consists of several different panels. Each panel has its own content consisting of different components. These components can either be created dynamically when the panel is shown or created statically and filled with data when the panel is shown.

Dynamic creation of a text field when tapping a button

```
var button = new Ext.Button({
  text: 'Dynamic creation',
  handler: function (btn, evt){
    panel1.add({
      xtype: 'textfield',
      label: 'Name:',
      value: 'Christian Veinfors'
    });
    mainPanel.setActiveItem(panel1); //switches to panel1
  }
});
```

Static creation of a text field. It is shown when tapping the button

```
var panel1 = new Ext.Panel({
  items: [{
    xtype: 'textfield',
    label: 'Name:',
    value: 'Christian Veinfors'
  }]
});

var button = new Ext.Button({
   text: 'Static example',
   handler: function (btn, evt){
      mainPanel.setActiveItem(panel1); //switches to panel1
   }
});
```

To start creating web applications with Sencha Touch, one needs to know data handling (as mentioned earlier with AJAX requests) and the concept of panel layout.

4.3 *jQuery Mobile*

jQuery Mobile is still in its early stages. In the moment of writing, the latest version available is Alpha 4.1. No date is yet decided when a stable version is to be released.

The archive available on the jQuery site contains the actual jQuery Mobile library file and a CSS-file. It also contains a folder with icons and images. One thing worth mentioning is that the regular jQuery library has to be downloaded as well.

There are plenty of demos and examples of how to use jQuery Mobile on the homepage but no real tutorials.

4.3.1 Getting started

To try out the framework a simple test application is created. It consists of a HTML file, the jQuery Mobile library file and the CSS file.

The HTML file consists of references to the library files and the stylesheet file.

Index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
    <title>Hello World</title>
    <script src="jquery-1.5.2.min.js" type="text/javascript"></script>
    <script src="jquery.mobile-1.0a4.1.min.js"
type="text/javascript"></script>
    <link href=" jquery.mobile-1.0a4.1.min.css" rel="stylesheet"
type="text/css" />
  </head>
  <body></body>
</html>
```

In contrast to Sencha Touch, the actual HTML code is written inside the body tag. Each page of the application is identified with a div element with the `data-role="page"` attribute.

Inside the div, any HTML markup can be used. To try out the framework, some other typical data-roles are used.

The content inside the body tag

```
<div data-role="page">
  <div data-role="header">
    <h1>Title text</h1>
  </div>
  <div data-role="content">
    <p>Hello World</p>
  </div>
  <div data-role="footer">
    <h4>Footer text</h4>
  </div>
</div>
```

The example code above creates a simple web application with a header, footer and some text in between. Worth mentioning is that not a single line of JavaScript has been written yet. The data-role attribute tells jQuery Mobile how the element should behave and be styled.

4.3.2 Developing the prototype

The first notable thing when starting the development is that in jQuery Mobile every page in the web application is its own HTML file.
This makes it possible to take advantage of the code behind approach of ASP.NET for handling external data, instead of using a HTTP handler or web service.

To be able to compare this prototype with the Sencha Touch prototype, some corresponding components and functions are used.

Again, a list is used to show some course participants.

Creation of list in the content div

```
<ul data-role="listview">
  <asp:Repeater ID="UserList" runat="server"
OnItemDataBound="UserList_ItemDataBound">
    <ItemTemplate>
      <li>
        <asp:HyperLink ID="LnkUser" runat="server"></asp:HyperLink>
      </li>
    </ItemTemplate>
  </asp:Repeater>
</ul>
```

When setting the data-role attribute to “listview” on an ul tag (unordered list), the jQuery Mobile framework automatically makes the list more mobile-friendly, both in functionality and style.

The users are fetched from the database in the code behind file. This data is binded to the ASP.NET repeater component, which will fill the list. What this component does is to fetch data one row at a time and construct a list of it.

Each list item is in the form of a hyperlink, which will navigate to the user information page. This page consists of labels and text fields to show the name and e-mail of the user. A button is also added to be able to save possible changes. The button functionality is, once again, taken care of in the code behind.

All these components are written in regular HTML markup and jQuery Mobile takes care of the styling automatically.

When tapping a user in the list a slide transition is used by default to display the user information page. In contrast to Sencha Touch, this transition does not need to be explicitly defined but is taken care of by the framework. Also, a back-button is added automatically to the header by jQuery Mobile.

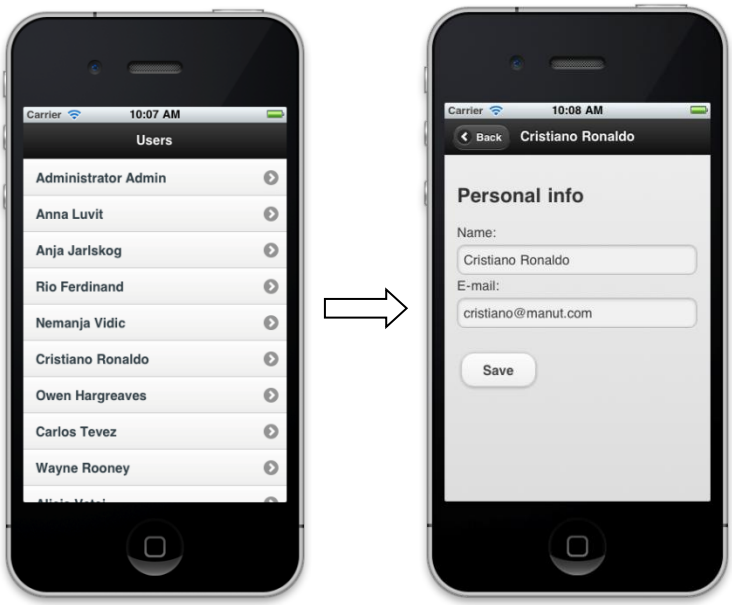

Figure 6: The jQuery Mobile prototype

4.3.3 Performance and appearance

The first thing noticed when trying out the prototype on some mobile devices is that the performance of jQuery Mobile is not as good as with Sencha Touch. Scrolling is less smooth and transitions are a little choppier, especially on Android devices.

As mentioned in chapter 4.2.3, the loading time is faster for jQuery Mobile than for Sencha Touch.

The standard appearance of the framework uses discrete colors. Black, grey and white are mostly used throughout the components. It makes the application look stylish and professional.

The theming in jQuery Mobile is made by adding the attribute *data-theme* to an optional HTML tag. The attribute can be set on entire pages but also on single components.

Example of changing a toolbar to theme b

```
<div data-role="header" data-theme="b">
  <h1>Page Title</h1>
</div>
```

There are 5 different themes available at the moment, called a, b, c, d and e.

Figure 7: the different themes available in jQuery Mobile

In contrast to Sencha Touch, there is no smooth way of creating custom themes in jQuery Mobile. The main CSS file can of course be modified but it takes a lot of work and trial and error to get things right. According to the jQuery Mobile blog, better support for theming will be a priority in the future [13].

The framework comes with around 20 icons that can be applied on elements such as buttons and lists. This is less than what Sencha Touch offers.

4.3.4 Functionality

As expected, all the regular components such as buttons, lists, form elements and dialogs can be utilized in the framework. One important component that jQuery Mobile lacks is a tab bar, which is commonly used on mobile devices.

Also, the toolbar component in the framework does not work as well as the one in the version of Sencha Touch. It does not have good support for fixed positioning which is one of the features one would expect from a toolbar.

There is no specific functionality for handling data in jQuery Mobile, but one has to use regular jQuery instead. This has support for AJAX and JSONP. Since jQuery also has a large plug-in library, there are many other possibilities for handling data exchange. This plug-in library can of course also be utilized for other uses in a jQuery Mobile application.

Example of fetching data with AJAX in jQuery

```
$.ajax({
  type: 'POST',
  url: 'Handler.ashx?function=GetUsers&courseID=5186',
  contentType: 'application/json; charset=utf-8',
  dataType: 'json'
})
```

4.3.5 Developing in the framework

Developing a web application in jQuery Mobile is not very different from developing a regular web site. For instance, each page in a mobile application is its own HTML file in contrast to a Sencha Touch application. A web developer can rather easily and quickly create something that feels like a mobile application.

As seen earlier, a typical jQuery Mobile application mainly consists of regular HTML elements. To utilize the jQuery Mobile functionality, attributes are added to these elements (for instance, the *data-role* attribute). Then the framework styles and adds some typical web application functionality such as transitions between pages.

The actual programming in the application is done with jQuery so for a developer experienced in that, it will be a small step to take to be able to make mobile web applications in jQuery Mobile.

4.4 *Titanium Mobile*

As mentioned earlier in this thesis, the code written in Titanium Mobile gets translated to the native code of the mobile device. Therefore, the expectations on applications created in this framework are quite high,

especially when it comes to performance.

To get started with the development, Titanium Developer needs to be installed. This is a development environment used to develop Titanium Mobile applications. It is also necessary to install the desired software development kit for each target platform. For instance, if one is to develop an application for Apple and Android devices both SDK's need to be installed.

4.4.1 Getting started

When creating a new mobile project in Titanium Developer, a test application is automatically generated. This simple application consists of a tab bar with two tabs. Each of these tabs has its own page. This can help developers getting started by looking at the generated source code.

In Titanium Developer, there are two choices of how to run the created application. Either in an emulator or on an actual device connected to the computer. When launching the application it is first compiled into native code and then installed on the device.

4.4.2 Performance

To get a fair view of the performance of Titanium Mobile, the Kitchen Sink application was tried out on a real Android device. The performance did not meet the high expectations. Transitions and the overall flow of the application were not as smooth as one would expect from a native application. Though, the start-up time is significantly better than with Sencha Touch and jQuery Mobile.

4.4.3 Difficulties

The most prominent difficulties and drawbacks of Titanium Mobile are testing and distribution. Each time a change is made to the application it has to be compiled and installed on the device or emulator. This is rather tedious compared to testing of a web based application.

The option of making the application executable through a web browser also disappears when using this framework. Instead the application has to be distributed through Apple App Store and Android Market.

The performance needed to be really good in order to outweigh the drawbacks of the framework. Since this was not the case, it was decided not to examine Titanium Mobile any further. Because of this, no prototype was created.

4.5 *PhoneGap*

PhoneGap is recommended by Sencha for deploying web applications. This makes it suitable to use the prototype developed in Sencha Touch to try out PhoneGap.

The cloud service provided by PhoneGap for building the application is used in this case. Hence, it is not necessary to install PhoneGap or any mobile SDK on the computer in order to use the tool.

4.5.1 Getting started

There is only one requirement that needs to be met for the web application to be able to use PhoneGap's build service. This is that the application contains an HTML file. In reality, no application consist of just that so in the typical case a zip archive is uploaded instead with all the necessary files. Once this is done, the build service will create installation files for all the different platforms that the developer can download and install.

4.5.2 Developing the prototype

To try out PhoneGaps' support for accessing native features of different mobile devices, the Sencha Touch prototype is expanded. The functionality added is the ability to capture photos with the camera and then upload them to a server.

PhoneGap code mixed with Sencha Touch code to access the gallery of a device

```
new Ext.Button({
  text: 'Upload photo',
  listeners: {
    click: {
      fn: function(){
        navigator.camera.getPicture(onPhotoURISuccess, fail1, {
          quality: 30,
          destinationType: navigator.camera.DestinationType.FILE URI,
          sourceType: navigator.camera.PictureSourceType.PHOTOLIBRARY
        });
      }
    }
  }
})
```

As seen in the example code above, it is not necessary to write much in order to add native features. Of course, this prototype has to be tested on a real mobile device with a built in camera since no emulators provide this.

Since all basically PhoneGap does is to wrap a web application, neither the performance nor appearance is affected. This only depends on the framework that the application is built with, for instance Sencha Touch.

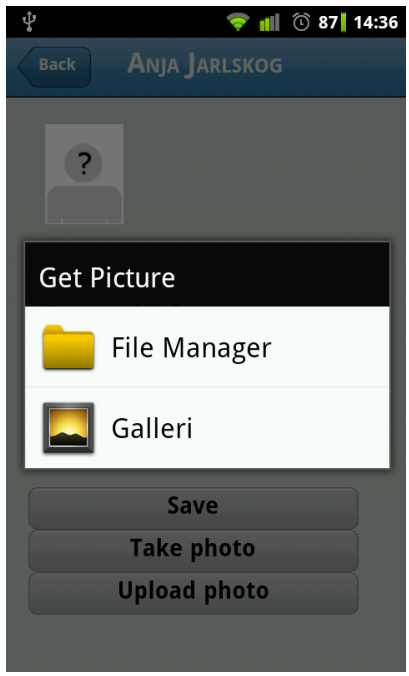

Figure 8: PhoneGap prototype when the upload button has been tapped

4.6 *Conclusions*

There are many aspects to consider when choosing the most appropriate solution for developing Luvit Mobile. The main focus has been much on features that can rather easily be measured or experienced visually. But in the end, it is still the general user experience that matters. Even if the application performs well and looks good it is not guaranteed that the end user will enjoy it.

As a developer it might be hard to find the balance between user experience and more measurable properties of the application. In the end, it will more or less be our subjective opinions on what is the best choice.

4.6.1 Performance

Performance is an important aspect of any application. When it comes to mobile applications, it is even more important due to limited hardware resources.

PhoneGap is excluded from this part of the comparison of the frameworks for apparent reasons. Sencha Touch and jQuery Mobile falls into one category while Titanium Mobile falls into another since it compiles the source code into native code. Therefore, the demands for Titanium are higher.

When it comes to start-up time, Titanium is clearly the best choice. When running Sencha Touch or jQuery Mobile applications from an external web server, the loading time is fairly slow. Sencha Touch is the slowest of the three.

When it comes to general flow and transitions in the applications, Titanium is a disappointment. It performs a little better than the other two frameworks but you would expect much more. Sencha Touch performs slightly better than jQuery Mobile.

4.6.2 Functionality

To get a fair assessment of the different frameworks in terms of functionality we will compare Titanium Mobile with Sencha Touch and jQuery Mobile when both are combined with PhoneGap.

Both PhoneGap and Titanium Mobile have support for the most common native features like camera, GPS and notifications. PhoneGap has support for more platforms, though, which is a great advantage.

When it comes to functionality of the user interface, both Titanium Mobile and Sencha Touch offers everything one might wish. jQuery Mobile is still a little behind but has potential, especially since it is only in an early alpha version.

Sencha Touch offers great possibilities for changing or customizing your own theme which jQuery Mobile still lacks.

4.6.3 Final decision

When taking all the aspects in this chapter in consideration, the final choice for developing Luvit Mobile is by using Sencha Touch together with PhoneGap. The functionality and performance of Sencha Touch is satisfying but the most important reason for choosing Sencha Touch is the overall experience of the framework. It has a steep learning curve but when overcoming this obstacle it is a delight to work with.

To fully utilize a mobile device and more of its functions, PhoneGap is the obvious choice to combine with Sencha Touch.

4.6.4 Tools

Since Sencha Touch is a rather new framework, the range of tools is quite limited. The only development environment with syntax support for Sencha Touch found was Aptana Studio 3. It worked reasonably satisfying and made the implementation a bit easier.

The web browser Google Chrome was used extensively during the development, both for its good support for Sencha Touch but also for the built-in developer tools [14]. The graphical debugger can be helpful when debugging JavaScript and the Elements panel for inspecting the DOM. The DOM can also be edited directly which can save a lot of time and makes the development a lot easier.

Another great tool that was used was Fiddler, which is a web debugger that logs the HTTP traffic [15]. This is very useful when debugging AJAX requests which is used through-out Luvit Mobile.

CHAPTER 5

5 Requirements specification

Before starting the development it is needed to know what has to be implemented. The requirements were developed iteratively. For instance, on the first meeting only a handful of requirements were decided to be implemented.

In the end, the requirements specification included the following features.

- **Authentication**: The user should be able to log in and log out in a safe way. The same credentials as for regular Luvit are to be used.
- **Course selection**: A user should be able to select a course that he/she already participates in.
- **Course contents**: As in regular Luvit, the user should to be able to access different course content. This can be HTML files, PDF files, picture files, links to other sites, tests and so on. The course administrator chooses what content are "mobile friendly" and thereby shown in the application.
- **User information:** A user should be able to view the other participants in a course. Also, a subset of a users' profile presentation from regular Luvit is to be shown such as e-mail, phone number and a profile picture. When pressing the e-mail or phone link the mobile device should open the e-mail application or call the number.
- **Message handling:** A user should be able to read and write messages to other course participants.
- **Bulletin board:** A user should be able to read news in a course that the administrator has published.
- **Course activities:** A user should be able to view different activities in the course. This can be when someone changes their personal presentation, when someone makes a personal status update and so on. A user should also be able to make their own status update and comment on others.

- **Calendar:** A user should be able to see the upcoming events in a course. This can both be course events or self-added events.
- **Assignments:** A user should be able to view all kind of information about the assignments in a course and also the history about handed in assignments.
- **Tracking:** A user should be able to view the different personal results in a course. This can be the general course status, status about assignments and status about assessments.

CHAPTER 6

6 Results

6.1 *Application structure of the front-end*

The design pattern used in the application is called Model-View-Controller (MVC). In the latest versions of Sencha Touch it is recommended to use this.

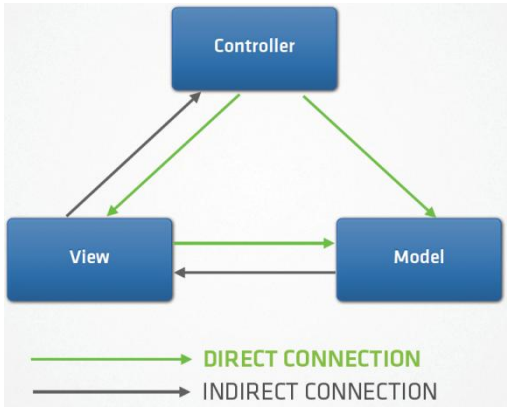

Figure 9: the concept of MVC [24]

The benefits of using the MVC pattern is that the codebase gets a good structure which is less complex and therefore easier to maintain and develop further [25].

It is often wise to separate data (*Model*) from the presentation layer (*View*). By doing this, one can for example re-organize the data without having to change the view and vice versa.

- **Model**: Manages the data handling of the application. It also decides in what form data is represented.
- **View:** Renders the data into a user interface. This is the part of the application that is visible to the user.
- **Controller:** When a user interacts with the application, the *Controller* instructs the *Model* and *View* with changes to be made.

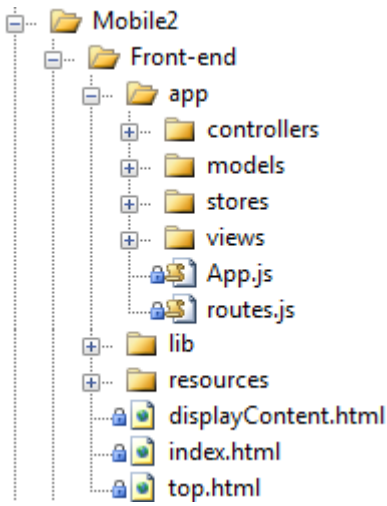

Figure 10: the structure of Luvit Mobile

As shown in Figure 10, the MVC pattern is used in the finished Luvit Mobile application. Worth mentioning is that a folder named *stores* also has been added which contains data structures that holds the data that the *Model* represents. Some code examples that have been simplified are shown below.

View: consists of a list component that shows *Title* and *Date* of the model

```
Luvit.views.Activity = Ext.extend(Ext.List, {
  store: Luvit.stores.Activity,
  itemTpl: '<p class="title">{Title}</p><p class="date">{Date}</p>',
  ui: 'round'
});
```

Model: decides how the store should be filled

```
Luvit.models.Activity = Ext.regModel('Luvit.models.Activity', {
  fields: [
    {name: 'Title', type: 'string'},
    {name: 'Date', type: 'string'}
  ]
});

function fillActivityStore() {
  // AJAX call to a web service. Fills the corresponding store with data.
}
```

Store

```
Luvit.stores.Activity = new Ext.data.Store({
  model: 'Luvit.models.Activity'
});
```

Controller: sets what view to be shown

```
Ext.regController("Activity", {
  show: function() {
    Ext.getCmp('backButton').show();
    Ext.getCmp('toolbar').setTitle('Activity');
    Luvit.views.viewport.setActiveItem(
      Luvit.views.Activity
    );
  }
});
```

6.2 *The back-end*

The back-end consists of .NET Web Services. The web services basically consist of web methods which get and set data from the database.

The objects that are returned and received by the web services are in the JSON format.

Below is a code example of a web method.

The object that is returned

```
public class Activity
{
  public string Title { get; set; }
  public string Date { get; set; }
}
```

The web method. Returns the Activity object in JSON format.

```
[System.Web.Services.WebMethod(EnableSession = true)]
[ResponseFormat = System.Web.Script.Services.ResponseFormat.Json)]
public Activity GetActivity()
{
  DataTable dt = GetData(); // Fetches information from the database
  var activity = new Activity();
  activity.Title = dt.Rows["title"].toString();
  activity.Date = dt.Rows["date"].toString();
  return activity;
}
```

In the figure below the software design for Luvit Mobile can be seen. In comparison with figure 2, the entire front-end has been replaced. Some parts from the back-end have been reused but are now in the form of webservices. The same database is still used.

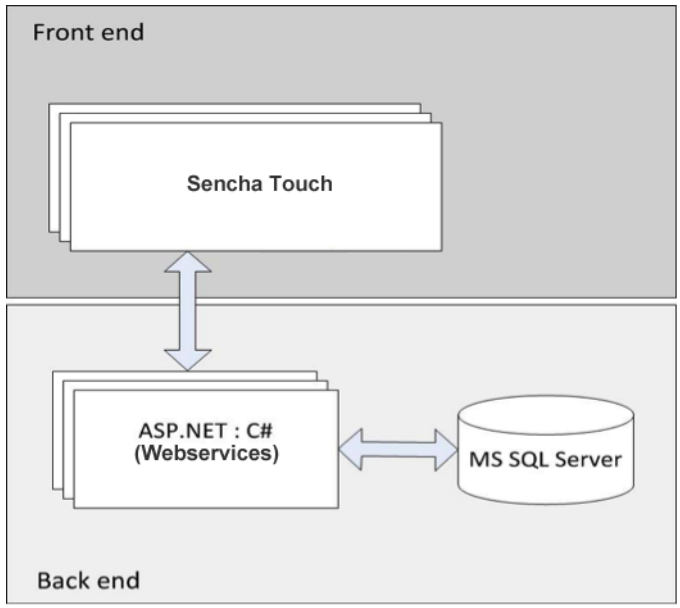

Figure 11: Software design of Luvit Mobile

6.3 *Encountered problems*

During all kind of development, problems occur. This is almost impossible to avoid. In this project a lot of minor problems emerged since Sencha Touch had not been used before by the developers of Luvit Mobile. Also, technologies like AJAX had not been used before.

One of the small problems encountered was how to handle the back button functionality. Some pages in the application could be reached from several places so one cannot just save the previous page in a history variable. This was solved with the *stack* data structure, which saved all the previous pages a user has visited. When going back, the latest visited page was popped from the stack and redirected the user to the right location.

On some places in the application the Sencha Touch components lacked some functionality that was needed to make the application work in a satisfying way. For instance, some small components were styled manually with the help of JavaScript and CSS.

6.3.1 Downloading on Android

One of the biggest problems that occurred was the lack of support for downloading files in regular Luvit to Android devices. The way that Luvit

is built, it streams out files to the client. Before the whole output stream had been sent to the client it always stopped for no apparent reason and only on Android devices.

After reading a lot of forums and development blogs a solution was found [16]. When streaming out the file, a content-type and filename had to be added to the HTTP header in a very specific way.

The content-type that was needed was *application/octet-stream.* The filename had to be surrounded with double quotation marks and the file extension MUST to be in upper case. After changing this, the file download worked perfectly on Android devices.

6.3.2 Embedding files in Sencha Touch

Another large problem that took a lot of time during the development was the limited support of embedding files in Sencha Touch. To make the application intuitive and easy to work with, the user should not have to leave it in order to view external files like HTML pages, images and PDF files.

The problem with just including these files directly in the Sencha Touch framework is that one loses the built-in ability of the web browser to interact with them. For example, if an image is too large to be fully displayed on the screen the user should be able to zoom out using pinch gestures. Sencha Touch can catch these kinds of events but the developer has to fully implement what should happen when using them. Since this works flawlessly in the browser, it would be better to utilize this instead of building the wheel again.

The simplest solution would be to just open the files in a new browser window. This is not a good solution for many reasons. The user loses the ability to easily navigate backwards in the application. Also, if the application is compiled with PhoneGap and the mobile device does not support multitasking (like older iPhones) the application has to be restarted.

The solution chosen for Luvit Mobile is a very special one but it gets the job done. The external file is opened in the same window, but a frame is added at the top of the window with a Sencha Touch styled navigation bar with a back button. By doing this, you can fully utilize the browsers'

functionality for zooming and scrolling but the user does not notice that he/she has left the application.

The user is actually redirected to an HTML page consisting of two frames. A path to the file that should be displayed is included in the query string[1], and the file is displayed in the bottom frame using an AJAX request. The query string also contains a variable that is used when the user taps the back button in the top frame to go back to the previous place in the application. Thanks to this solution, the requirement on course contents mentioned in chapter 5 was fulfilled.

Figure 12: Example of displaying an image file in Luvit Mobile

6.4 *Performance*

In order to make the user experience of the application as good as possible it is of great importance to maximize the performance. To make this in an efficient way it is important to identify possible bottlenecks. According to

[1] The query string is the part of the URL used to pass data to a web application (marked in bold below)
Example: http://grade.com?**data=hello**

Tommy Maintz, Lead Developer of Sencha Touch, the DOM size is the most important thing to consider when optimizing web applications [17]. When creating the different pages in Luvit Mobile, this has been taken into account to make the flow of the application as smooth as possible.

To optimize the performance even more, event listeners on buttons was created only when needed and removed otherwise. This led to a great performance boost for this application.

Another thing that increased the general flow of the application was to avoid showing the loading animation at the same time as a page transition. Instead the animation was only showed during the AJAX request. When this call was finished, the animation was removed and then the page transition was made and the data from the request was shown.

Since the finished application consisted of about 70 JavaScript files with about 2500 lines of code it was needed to merge these files into one file to make the loading time better. Also, a minification was done to make this one file even smaller. This made the file size around 35 percent smaller. This minification was done with *jsbuilder 3* and *ycompressor* that came bundled with Sencha Touch. On older phones, the loading times decreased by over 50 percent.

6.5 *Final result*

The requirements in chapter 5 were all successfully implemented. Below are some screenshots of the final application.

Figure 13: the main menu of Luvit Mobile and the user profile view

CHAPTER 8

7 Conclusions

7.1 *Discussion*

After having been working with Sencha Touch for a fair amount of time now, it is easier to draw conclusions regarding advantages/drawbacks about both the specific framework but also about mobile web development in general.

The biggest concern in the beginning of the development of Luvit Mobile was the steep learning curve, mentioned earlier in this thesis. After some time you get used to the unusual syntax and how it works in general. When working with the basic components and creating simple applications like the demo applications on the Sencha site [1] the development is a smooth ride.

When it comes to creating a large application with more specific demands, both in functionality and styling, it gets tougher. This depends a lot on the immaturity of the framework and sometimes the lack of documentation. Thus one has to use a lot of special solutions that can be time consuming.

Another concern was the performance compared to native applications. As Luvit Mobile grew bigger the performance seemed to get worse and we started to get doubts about the framework. Although, after making the optimizations mentioned in chapter 6.4 the application started to feel smooth and fast loading.

When trying out the application on actual mobile devices (as opposed to simulators), it became more obvious that the optimization changes had made a great impact on the performance. On older devices the application does not feel as smooth as one would hope, though.

[1] http://www.sencha.com/products/touch/demos/

When it comes to appearance, there were no problems at all to make it look and feel like a professional native application. This result was expected after the investigations in earlier chapters in this thesis.

7.1.1 Native versus web based

So, what are our opinions about Sencha Touch versus a native solution?

When it comes to Luvit Mobile, most of the advantages will probably occur in the future when the application needs to be maintained. It is so much less time consuming having to update one web based application instead of several native ones. Since Grade releases their product Luvit in different customized versions to different customers, this solution with Sencha Touch and PhoneGap will work very well.

In our own opinion, Sencha Touch is a well-suited solution for Luvit Mobile. This is because in Luvit the users mostly just read and write different kind of information.

For applications with more advanced graphics, such as games, and applications which utilize much of the devices' hardware it is wiser to choose a native solution. This is because with Sencha Touch you run the application in a web browser which simply gives worse performance than an application that runs closer to the hardware.

Of course, one can always access hardware functionality with tools like PhoneGap but it can only utilize a limited set of functions of the device and not all of them like with a native solution.

As mentioned above, this web based solution is probably the better choice for Grade as a company but from our point of view we would prefer a little better performance on older devices. Also, we are more comfortable working with an object oriented programming language instead of a scripting language.

In conclusion, we can really recommend Sencha Touch and PhoneGap but if your applications performance needs to be flawless, even on older devices, a native solution is the better choice.

These web based solutions gets better every day and the performance of mobile devices is also increasing in a fast pace so the future looks bright.

7.2 *Future work*

In the future, Luvit Mobile will probably grow larger with more functionality. Thanks to the good structure in the code base with the use of the MVC pattern the expansion will be relatively simple.

At the moment, Luvit Mobile only uses PhoneGap to wrap the application making it possible to install as a native one. As mentioned in chapter 4.5 the tool can be utilized to do much more and will probably also do so in the future.

References

[1] (2011, Mar.) Utbildningssystemet LUVIT. [Online]. http://www.grade.se/luvit-lms.aspx

[2] Grade. (2011, Mar.) Grade. [Online]. http://www.grade.se

[3] Stefan Larsson, Grade. Interview, Mar. 2011.

[4] Jonathan Stark, *Building iPhone Apps with HTML CSS and JavaScript*., 2010.

[5] (2011, Mar.) Sencha Touch. [Online]. http://www.sencha.com/products/touch/

[6] (2011, Apr.) jQuery Mobile. [Online]. http://jquerymobile.com/

[7] (2011, Apr.) Titanium Mobile. [Online]. http://www.appcelerator.com/products/titanium-mobile-application-development/

[8] (2011, Apr.) PhoneGap. [Online]. http://www.phonegap.com/story

[9] (2011, Apr.) Android Issues. [Online]. http://code.google.com/p/android/issues/detail?id=6914

[10] (2011, Apr.) Sencha Touch Theming. [Online]. http://www.sencha.com/blog/an-introduction-to-theming-sencha-touch

[11] (2011, Apr.) JSONP. [Online]. http://bob.pythonmac.org/archives/2005/12/05/remote-json-jsonp/

[12] (2011, Apr.) YQL. [Online]. http://developer.yahoo.com/yql/

[13] (2011, Apr.) jQuery Mobile Blog. [Online]. http://jquerymobile.com/blog/2011/03/31/jquery-mobile-alpha-4-released/

[14] (2011, Aug.) Chrome Developer Tools. [Online]. http://code.google.com/intl/sv-SE/chrome/devtools/docs/console.html

[15] (2011, Aug.) Fiddler. [Online]. http://www.fiddler2.com/fiddler2/

[16] (2011, Aug.) digiblog.de. [Online]. http://digiblog.de/2011/04/19/android-and-the-download-file-headers/

[17] (2011, Aug.) Performance Optimization for Sencha Touch. [Online]. http://vimeo.com/17699976

[18] (2011, Mar.) W3C. [Online]. http://www.w3.org/DOM

[19] (2011, Mar.) WebKit. [Online]. http://www.webkit.org

[20] (2011, Mar.) SASS. [Online]. http://sass-lang.com/

[21] (2011, Apr.) Wikipedia. [Online].

http://en.wikipedia.org/wiki/Minification_(programming)

[22] (2011, Apr.) IASA. [Online]. http://www.iasa.se/?p=267

[23] (2011, Apr.) Adaptive Path. [Online]. http://www.adaptivepath.com/ideas/e000385

[24] (2011, Aug.) Slideshare [Online]. http://www.slideshare.net/senchainc/structuring-your-sencha-touch-application

[25] Leff, A., Rayfield, J.T, "Web-application development using the Model/View/Controller design pattern". Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International, pages 118-127.

List of Acronyms

LMS	Learning Management System
DOM	Document Object Model
HTML	Hyper Text Markup Language
CSS	Cascading Style Sheets
SASS	Syntactically Awesome Stylesheets
SDK	Software Development Kit
GPU	Graphics Processing Unit
API	Application Programming Interface
MVC	Model View Controller